Université Paris-Est
École doctorale MSTIC

THÈSE

pour l'obtention du grade de Docteur de l'Université Paris-Est
en spécialité Informatique
présentée et soutenue publiquement par

EDWIN CARLINET

le 27 novembre 2015

UN ARBRE DES FORMES POUR DES IMAGES MULTI-VARIÉES
A TREE OF SHAPES FOR MULTIVARIATE IMAGES

Jury :

Président :	Ludovic Macaire, Pr., Université Lille 1
Rapporteurs :	Coloma Ballester, Pr., Universitat Pompeu Fabra Philippe Salembier, Pr., Universitat Politècnica de Catalunya Ludovic Macaire, Pr., Université Lille 1
Examinateurs :	Jesús Angulo, Pr., MINES ParisTech Pascal Monasse, Pr., École des Ponts ParisTech
Directeurs de thése :	Jean Serra, Pr., ESIEE-Paris Thierry Géraud, Pr., EPITA

ACKNOWLEDGEMENTS

I would like to thank Coloma Ballester, Philippe Salembier who kindly accepted to review this dissertation. I also thank Jesús Angulo and Pascal Monasse who agreed to be members of the jury. Special thanks go to Ludovic Macaire who accepted to be both reviewer and president of the jury.

I would also like to thank Jean Serra, one of my thesis supervisor, for the advise he gave me all through these three years. We have not been in touch a lot, but the research tracks and the ideas provided were very helpful for pursuing this thesis.

I am very grateful (the word is definitely not strong enough) to Thierry Géraud, my other supervisor, for his support, his ideas, his jokes, his patience, his *everything* he gave me. He has not only been my supervisor for just 3 years, he was also my "mentor" since my 3rd year of Epita, when I was still an innocent young student, barely knowing the programming basics... He taught me everything I know about advanced programming and image processing: from those crazy things we can do in C++, to the fact that Mathematical Morphology can work in practice (yes, it can...). He found the right way to supervise: mixing rush periods for publishing and IP challenges, and slack periods to keep me alive! Long story short, I am endless grateful to him and I hope we will have the opportunity to work together in the future.

Concerning the LRDE, I would like to thank all the people I have been working more or less closely with: Jim, who has reviewed this dissertation (just hope it was not such painful); Akim and Alexandre, for our general talks about programming and computer science, they probably are the most skilled people I ever met; Yongchao, Jonathan... for our talks about IP; Olivier, who gave me the opportunity to work on this thesis; and all my other colleagues, Daniela, Didier, Reda, Clément, Ala, Ana, for everything. A particular thank goes to Étienne and Myriam. You guys probably have dropped my productivity because of long coffee breaks, endless Tuesday jokes, long talks, etc., but it was worth it. You helped me a lot in putting up with the writing phase of my thesis... so thanks!

Then, I would like to thank all my friends. Shaoner, Coquart, Natacha, Thomas, Edouard, Maxime, Kevin, "flatmate" Julien, I met you at Epita quite a long time ago, but we succeed in keeping in touch. I cannot tell you the pleasure it is when I meet you guys to have a drink. Thank you all! I also have a think for all my friends I met in high school, and Groningen. It is getting harder to stay in touch, but Florian, Romain, Hélène, "Eloyes's trio", Carlos, Giulia, Sara, Erik, etc. I have not forgotten any of you.

Finally, I really thank my family, especially my parents who have always believed in me and my "abilities", and did everything so that I could achieve my goals. This includes giving priority to my needs before theirs... for that, thanks!

I could not finish without some words to my *sunshine* (even if those words are clumsy and definitely the most difficult to write). You think that I help you a lot but this is nothing compared to what you bring me and I probably need you more than you need me. “I am wearing the smile you give me” all day long and if I have been able to support this critical period, this is mostly thanks to the moments you share with me. The drinks we have together, our squash games, our after-squash games, our resting moments in the park, the way you make feel *high all the time*, etc. for all of these and everything else, *thank you very much sunshine*.

ABSTRACT

Nowadays, the demand for multi-scale and region-based analysis in many computer vision and pattern recognition applications is obvious. No one would consider a pixel-based approach as a good candidate to solve such problems. To meet this need, the Mathematical Morphology (MM) framework has supplied region-based hierarchical representations of images such as the Tree of Shapes (ToS). The ToS represents the image in terms of a tree of the inclusion of its level-lines. The ToS is thus self-dual and contrast-change invariant which make it well-adapted for high-level image processing. Yet, it is only defined on grayscale images and most attempts to extend it on multivariate images - *e.g.* by imposing an "arbitrary" total ordering - are not satisfactory.

In this dissertation, we present the Multivariate Tree of Shapes (MToS) as a novel approach to extend the grayscale ToS on multivariate images. This representation is a mix of the ToS's computed marginally on each channel of the image; it aims at merging the marginal shapes in a "sensible" way by preserving the maximum number of inclusion. The method proposed has theoretical foundations expressing the ToS in terms of a topographic map of the curvilinear total variation computed from the image border; which has allowed its extension on multivariate data. In addition, the MToS features similar properties as the grayscale ToS, the most important one being its invariance to any marginal change of contrast and any marginal inversion of contrast (a somewhat "self-duality" in the multidimensional case).

As the need for efficient image processing techniques is obvious regarding the larger and larger amount of data to process, we propose an efficient algorithm that can build the MToS in quasi-linear time w.r.t. the number of pixels and quadratic w.r.t. the number of channels. We also propose tree-based processing algorithms to demonstrate in practice, that the MToS is a versatile, easy-to-use, and efficient structure.

Eventually, to validate the soundness of our approach, we propose some experiments testing the robustness of the structure to non-relevant components (*e.g.* with noise or with low dynamics) and we show that such defaults do not affect the overall structure of the MToS. In addition, we propose many real-case applications using the MToS. Many of them are just a slight modification of methods employing the "regular" ToS and adapted to our new structure. For example, we successfully use the MToS for image filtering, image simplification, image segmentation, image classification and object detection. From these applications, we show that the MToS generally outperforms its ToS-based counterpart, demonstrating the potential of our approach.

Keywords: Tree of Shapes, connected operators, mathematical morphology, level sets, hierarchies, multivariate images, color images.

RÉSUMÉ

De nombreuses applications issues de la *vision par ordinateur* et de la *reconnaissance des formes* requièrent une analyse de l'image multi-échelle basée sur ses régions. De nos jours, personne ne considérerait une approche orientée « pixel » comme une solution viable pour traiter ce genre de problèmes. Pour répondre à cette demande, la Morphologie Mathématique a fourni des représentations hiérarchiques des régions de l'image telles que l'Arbre des Formes (AdF). L'AdF représente l'image par un arbre d'inclusion de ses lignes de niveaux. L'AdF est ainsi auto-dual et invariant au changement de contraste, ce qui fait de lui une structure bien adaptée au traitements d'images de haut niveau.

Néanmoins, il est seulement défini aux images en niveaux de gris et la plupart des tentatives d'extension aux images multivariées (*e.g.* en imposant un ordre total « arbitraire ») ne sont pas satisfaisantes.

Dans ce manuscrit, nous présentons une nouvelle approche pour étendre l'AdF *scalaire* au images multivariées : l'Arbre des Formes Multivarié (AdFM). Cette représentation est une « fusion » des AdFs calculés marginalement sur chaque composante de l'images. On vise à fusionner les *formes* marginales de manière « sensée » en préservant un nombre maximal d'inclusion. La méthode proposée a des fondements théoriques qui consistent en l'expression de l'AdF par une carte topographique de la variation totale curvilinéaire depuis la bordure de l'image. C'est cette reformulation qui a permis l'extension de l'AdF aux données multivariées. De plus, l'AdFM partage des propriétés similaires avec l'AdF scalaire ; la plus importante étant son invariance à tout changement ou inversion de contraste marginal (une sorte d'*auto-dualité* dans le cas multidimensionnel).

Puisqu'il est évident que, vis-à-vis du nombre sans cesse croissant de données à traiter, nous ayons besoin de techniques *rapides* de traitement d'images, nous proposons un algorithme efficace qui permet de construire l'AdF en temps quasi-linéaire vis-à-vis du nombre de pixels et quadratique vis-à-vis du nombre de composantes. Nous proposons également des algorithmes permettant de manipuler l'arbre, montrant ainsi que, en pratique, l'AdFM est une structure facile à manipuler, polyvalente, et efficace.

Finalement, pour valider la pertinence de notre approche, nous proposons quelques expériences testant la robustesse de notre structure aux composantes non-pertinentes (*e.g.* avec du bruit ou à faible dynamique) et nous montrons que ces défauts n'affectent pas la structure globale de l'AdFM. De plus, nous proposons des applications concrètes utilisant l'AdFM. Certaines sont juste des modifications mineures aux méthodes employant d'ores et déjà l'AdF scalaire mais adaptées à notre nouvelle structure. Par exemple, nous utilisons l'AdFM à des fins de filtrage, segmentation, classification et de détection d'objet. De ces applications, nous montrons ainsi que les méthodes basées sur l'AdFM surpassent généralement leur analogue basé sur l'AdF, démontrant ainsi le potentiel de notre approche.

Keywords : Arbre des Formes, opérateurs connexes, morphologie mathématique, ensemble de niveaux, hiérarchies, images multivariées, images couleur.

RÉSUMÉ LONG

RÉSUMÉ

L'Arbre des Formes (AdF) est un arbre morphologique qui fournit une représentation hiérarchique de l'image auto-duale et invariante par changement de contraste. De ce fait, il est adapté à de nombreuses applications de traitement d'images. Néanmoins, on se heurte à des problèmes avec l'AdF lorsqu'on doit traiter des images couleurs car sa définition tient uniquement en niveaux de gris. Les solutions les plus courantes sont alors d'effectuer un traitement composante par composante (marginal) ou d'imposer un ordre total. Ces solutions ne sont généralement pas satisfaisantes et font survenir des problèmes (des artefacts de couleur, des pertes de propriétés...) Nous insistons ici sur la nécessité d'une représentation à la fois auto-duale et invariante par changement de contraste et nous proposons une méthode qui construit un arbre unique, l'Arbre des Formes Multivarié (AdFM), en fusionnant des formes issues des composantes marginales tout en préservant les propriétés intrinsèques de l'arbre. Cette méthode s'affranchit de tout relation d'ordre totale en utilisant uniquement la relation d'inclusion entre les formes et en effectuant une fusion dans l'espace des formes. Finalement, nous montrerons la pertinence de notre méthode et de la structure en les illustrant à travers diverses applications de vision par ordinateur.

1 INTRODUCTION

L'AdF [32, 28] est une représentation hiérarchique de l'image témoignant de l'inclusion de ses lignes de niveaux. L'efficacité de cette structure pour le traitement d'images réside dans ses propriétés. D'abord, c'est une représentation morphologique (donc invariante par changement de contraste) basée sur l'inclusion des composantes connexes de l'image obtenues à différents niveaux de seuillage. En conséquent, un filtrage basique de cet arbre est un filtre connexe, c'est-à-dire un filtre qui ne déplace pas les contours des objets mais se contente de supprimer ou préserver certains d'entre eux [112]. Ensuite, non seulement cette structure est invariante par changement de contraste global mais elle l'est aussi *localement* [21]. Cette propriété est très importante dans de nombreuses applications de vision par ordinateur où la robustesse au changement d'illumination est un véritable challenge, e.g. pour la mise en correspondance de scènes ou la reconnaissance d'objets. La fig. 1d montre cette invariance en simulant un changement d'illumination directement avec l'AdF, on a donc la même représentation arborescente sur cette image que sur l'image d'origine montrée sur la fig. 1b. Enfin, au delà de son invariance au changement de contraste, l'AdF est aussi une représentation auto-duale de l'image. Cette propriété est fondamentale dans un contexte où les structures peuvent apparaître à la fois sur un fond

(a)

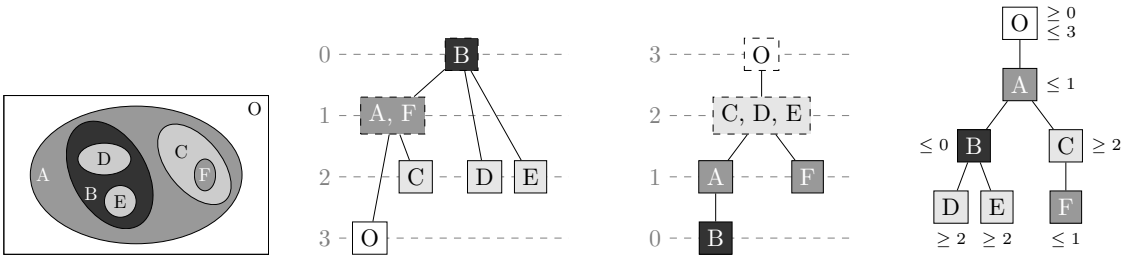
(b) (c) (d) (e)

FIGURE 1: (a) A propos du besoin d'invariance au changement et inversion de contraste (quelques exemples de changement et/ou inversion de contraste sur Lena). (b) Image originale. (c) Changement et/ou inversion de contraste indépendant sur chaque canal de (b). (d) Changement de contraste local sur (b) (simulation d'un changement d'illumination). (e) Quelques « lignes de niveaux » de l'Arbre des Formes Multivarié (AdFM). Les trois images (b), (c), (d) ont le même arbre des formes dont les lignes de niveaux sont montrées sur (e).

plus clair ou plus foncé. C'est pourquoi les opérateurs auto-duaux sont particulièrement bien adaptés pour le traitement des images où l'organisation du contenu n'est pas connu a priori. Alors que des opérateurs morphologiques tentent d'être auto-duaux (e.g. les filtres séquentiels alternés) en combinant des filtres extensifs et non-extensifs, certains dépendent en fait de l'ordre d'application (i.e. de quel filtre est appliqué en premier). Les filtres auto-duaux ont la capacité de traiter réellement de manière symétrique les objets foncés et clairs au même titre [53, 117] (voir la fig. 1c).

Malgré ces puissantes propriétés, l'AdF est encore largement sous-exploité, même si quelques auteurs l'ont d'ores et déjà utilisé efficacement dans des applications de traitement d'images et de vision par ordinateur. Dans [43, 13, 139, 142], les auteurs utilisent une approche par optimisation d'énergie sur la hiérarchie de l'AdF pour la simplification et la segmentation d'images en sélectionnant les lignes de niveaux significatives. D'autres applications incluent la segmentation de vaisseaux sanguins [144], le réglage d'images [28] et la mise en correspondance de scènes en étendant les MSER aux « Maximally Stable Shapes »[28] et aux « Tree-Based Morse Regions » [143].

Alors que l'AdF est bien défini sur les images à niveaux de gris, cela se complique sur les données multivariées. En effet, comme la plupart des arbres morphologiques (*e.g.* le min- et max-tree), l'AdF repose sur une relation d'ordre sur les valeurs qui doit être totale.



(a) Image originale. (b) Max-tree. (c) Min-tree. (d) AdF.

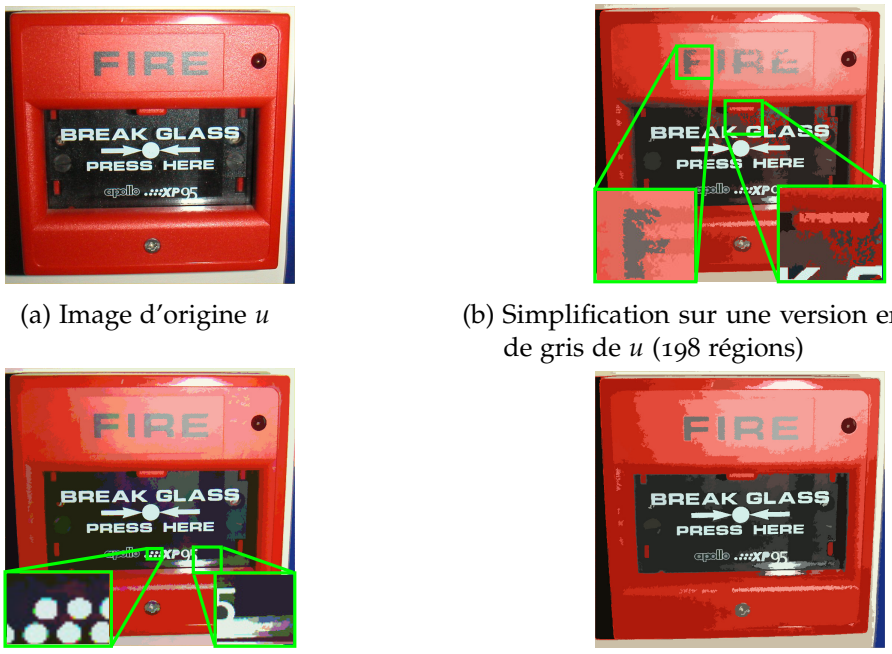
FIGURE 2: Représentations arborescentes basées sur le principe de décomposition.

Dans le cas contraire, les composantes connexes de coupes inférieures et supérieures se chevauchent et l'arbre d'inclusion est mal formé. Pour résoudre ce problème, on s'est le plus souvent concentré à définir un ordre total sur les données multivariées. Cependant, de notre point de vue, le concept le plus important dans les arbres morphologiques reste l'inclusion des formes. Par conséquent, on introduit une nouvelle approche qui s'affranchit de la nécessité d'un ordre total mais essaye de construire un ensemble de formes qui ne se chevauchent pas à partir d'un ensemble de formes quelconque en se basant uniquement sur la relation d'inclusion.

2 L'ARBRE DES FORMES ET LES PROBLÈMES LIÉS À LA COULEUR

2.1 *L'arbre des formes : définition et propriétés*

Soit $u : \Omega \to E$, une image définie sur un domaine Ω et prenant ses valeurs dans E muni d'une relation d'ordre $\leq$. Les lignes de niveaux de u sont la collection des ensembles de points $\{x \mid u(x) = \lambda, \lambda \in E\}$. En utilisant la représentation de [1], on peut s'assurer que ces ensembles forment des courbes fermées lorsque $\leq$ est un ordre total. En fait, l'ordre doit être total dû à la définition des lignes de niveaux en termes de contours de coupes inférieures ou supérieures. Soit $[u \leq \lambda]$ (resp. $[u \geq \lambda]$) une coupe inférieure (resp. supérieure) de u définie par $[u \leq \lambda] = \{x,\, u(x) \leq \lambda\}$. On note $\mathcal{CC}(X), X \in \mathcal{P}(E)$ l'ensemble des composantes connexes de X. Si $\leq$ est total, deux composantes connexes $X, Y \in \mathcal{CC}([u < \lambda])$ sont disjointes ou incluses. L'ensemble $\mathcal{CC}([u < \lambda])$ muni de la relation d'inclusion est un arbre appelé *min-tree* et son arbre dual, défini sur les coupes supérieures, est le *max-tree* (voir les figs. 2b and 2c). Soit l'opérateur de bouchage de trou $\mathcal{H}$, on appelle une *forme* un élément de $\mathcal{S} = \mathcal{H}([u \leq \lambda]) \cup \mathcal{H}([u \geq \lambda])$. Si $\leq$ est total, là encore deux formes seront disjointes ou incluses et donc la couverture de $(\mathcal{S}, \subseteq)$ forme un arbre : *l'Arbre des Formes (AdF)* (voir la fig. 2d). Dans le reste de ce papier, par abus de notation, on considère implicitement la couverture de $(\mathcal{S}, \subseteq)$ lorsqu'on écrit $(\mathcal{S}, \subseteq)$. Notons aussi que l'AdF traduit à la fois l'inclusion des formes et l'inclusion des lignes de niveaux puisque celles-ci sont les frontières des formes. Enfin, sans perte de généralité,



(a) Image d'origine u

(b) Simplification sur une version en niveaux de gris de u (198 régions)

(c) Simplification avec un traitement marginal (123 + 141 + 136 régions)

(d) Simplification avec notre approche (158 régions)

FIGURE 3: Les problèmes liés à la simplification avec les approches « standard » pour traiter la couleur. (b) montre le problème de fuite lorsque la luminance n'est suffisante pour obtenir complètement l'information géométrique. (c) montre le problème de fausses couleurs dues au traitement marginal. (d) Notre méthode récupère correctement le contenu principal de l'image sans introduire d'artefact visuel.

on considère $E = \mathbb{R}^n$ tout au long de ce papier et on notera u pour les images scalaires ($n = 1$) et $\mathbf{u}$ pour les images multivariées.

2.2 *Le problème des données multivariées : solutions standard et travaux connexes*

Les définitions précédentes des lignes de niveaux (en termes d'ensemble d'iso-niveau ou en tant que contour de formes) sont toutes deux mal formées en présence d'un ordre partiel. En effet, les ensembles d'iso-niveau ne forment pas des courbes fermées et les formes issues des coupes inférieures et supérieures peuvent se chevaucher, *i.e.*, $(\mathcal{S}, \subseteq)$ forme un graphe. Un moyen largement utilisé mais peu acceptable de résoudre le problème est de traiter l'image en niveaux de gris uniquement (e.g. la luminance). Cette approche reste plausible si on prétend que l'information géométrique est principalement contenue par la luminance [29]. Cependant, il n'est pas rare de rencontrer des images où les contours des objets n'existent qu'à travers la composante chromatique (notamment sur les images de document ou de synthèse). Ces exemples contredisent cette supposition

et montrent que la chrominance contient également l'information géométrique (voir la fig. 3b).

Une autre solution communément employée est le traitement de l'image canal par canal, puis la recombinaison des résultats individuels. Le traitement marginal est sujet au problème connu des fausses couleurs puisqu'il permet de créer des couleurs qui n'étaient pas présentes dans l'image d'origine. Les fausses couleurs peuvent être un problème ou pas (par exemple si les fausses couleurs sont suffisamment proches des couleurs d'origines pour ne pas être distinguées), mais dans le cas de la simplification, il produit des artéfacts de couleurs indésirables comme montrés sur la fig. 3c. Aussi, le traitement marginal produit au final plusieurs arbres (un pour chaque canal de l'image) alors que nous souhaitons obtenir une structure *unique* de l'image. Dans [2], nous avons proposé des idées préliminaires sur la manière d'obtenir un arbre unique à partir de plusieurs arbres.

Puisque le problème des formes qui se chevauchent est dû à l'ordre partiel des couleurs, des auteurs ont tenté d'imposer arbitrairement un ordre ou un pré-ordre *total*. Ils diffèrent en le fait qu'un nœud de l'arbre puisse être associé à une ou plusieurs valeurs. La manière d'ordonner un espace multivarié a été largement étudiée pour étendre les opérateurs morphologiques. [14] classifie les méthodes en quatre groupes : les ordres marginaux (M-ordering), les ordres conditionnels (C-ordering), les ordres partiels (P-ordering) et les ordres réduits (R-ordering). Alors que la première classe mène à un ordre partiel, les trois autres donnent un ordre ou un préordre total. Les ordres conditionnels visent à organiser les vecteurs en donnant des priorités à certaines (ou toutes) composantes du vecteur. L'ordre lexicographique, bien connu, appartient à cette classe et est la méthode la plus commune pour étendre l'AdF aux couleurs [36]. Les ordres réduits visent à projeter des données vectorielles sur un espace réel à travers une fonction de rang. Les fonctions de rang bien connues sont par exemple la norme l_1, la luminance dans un espace couleur donné, ou la distance à un ensemble de couleurs de référence. Elles ont été appliquées dans [123, 104, 89] pour la compression d'image et pour la détection d'objets astronomiques en utilisant les min ou max-trees mais l'idée est transposable pour l'AdF [3]. Des stratégies plus avancées ont été conçues pour donner naissance à un ordre total plus « sensé » où celui-ci dépend du contenu de l'image. Velasco-Forero et Angulo [127, 126] utilisent l'apprentissage automatique pour obtenir un P-ordering basé sur le partitionnement de l'espace, puis établissent une distance entre ces regroupements. Dans [74], l'apprentissage de variétés est utilisé pour inférer une fonction de rang sur les couleurs et dans [72], un ordre total est calculé localement sur une fenêtre spatiale glissante. Une liste non-exhaustive pour inférer un ordre total sur les données multivariées peut être trouvée dans [9].

Une autre approche introduite par Passat et Naegel [102] utilise directement l'ordre partiel des valeurs et manipule la structure sous-jacente qui est un graphe. Le graphe des composantes est encore à l'état de développement mais a montré des résultats prometteurs dans le cadre de filtrage [90]. Néanmoins, le graphe des composantes fait face à un problème de complexité algorithmique qui oblige les auteurs à effectuer le

filtrage localement. De ce fait, le graphe des composantes est à l'heure actuelle non adapté si l'on veut une représentation de l'image entière.

Dans [2], nous avons introduit une nouvelle approche où, au lieu d'essayer d'imposer un ordre total sur les valeurs, nous calculons marginalement les AdFs et les fusionnons en un arbre unique. Le procédure de fusion ne dépend pas d'un ordre total sur les valeurs mais sur des propriétés calculées dans l'espace des formes. Néanmoins, la stratégie de fusion proposée dans ce papier souffre d'un manque de « cohérence » car elle fusionne ensemble des formes sans liaisons apparentes. Dans [3], inspiré par le travail de PASSAT et NAEGEL [102], nous avons proposé le graphe des formes qui fusionne les AdFs marginaux en une seule structure et de façon efficace. Nous avons montré que cette structure avait un fort potentiel comparée aux approches standard qui imposent un ordre total. Cependant, la méthode construit un graphe, ce qui est une limitation puisque nous ne pouvons plus utiliser les outils fournis par l'état de l'art sur les arbres de composantes (filtrage, détection d'objets, méthodes de segmentation...) Le travail présenté ici peut être vu comme la poursuite des idées introduites dans [2] et [3] puisque le Graphe des Formes (GdF) est utilisé comme une représentation intermédiaire pour extraire un arbre unique depuis les formes issues des AdFs marginaux.

3 L'ARBRE DES FORMES COULEUR

3.1 *Contraintes et propriétés de l'arbre des formes couleur*

Commençons d'abord par relâcher la définition de *forme*. Une *forme* X est une composante connexe de Ω sans trous (*i.e.*telle que $\mathcal{H}(X) = X$). Soit une famille d'ensemble de formes $\mathcal{M} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ où chaque élément $(\mathcal{S}_i, \subseteq)$ forme un arbre. On note $\mathcal{S} = \bigcup \mathcal{S}_i$ l'ensemble initial de formes. Notons que $(\mathcal{S}, \subseteq)$ ne forme généralement pas un arbre mais un graphe puisque des formes peuvent se chevaucher. On cherche a définir un nouvel ensemble de formes $\mathcal{S}'$ tel que deux formes soient disjointes ou incluses. On ne se limite pas aux ensembles $\mathcal{S}' \subseteq \mathcal{S}$. En d'autres termes, on autorise la méthode à créer de nouvelles formes qui n'étaient pas présentes dans l'ensemble de formes d'origine. On note $T(\mathbf{u}) : \Omega^{\mathbb{R}^n} \to (\mathcal{P}(\mathcal{P}(\Omega)), \subseteq)$, le processus qui construit un arbre des formes $(\mathcal{S}', \subseteq)$ à partir d'une image $\mathbf{u} \in \Omega^{\mathbb{R}^n}$.

Plus formellement, on souhaite que la méthode T produise $(\mathcal{S}', \subseteq)$ avec les propriétés suivantes :

- *(R1) La couverture du domaine* $(\bigcup_{X \in \mathcal{S}'} X) = \Omega$ (tout point appartient à au moins une forme)
- *(R2) Une structure arborescente* $\forall X, Y \in \mathcal{S}'$, soit $X \cap Y = \emptyset$ ou $X \subseteq Y$ ou $Y \subseteq X$ (deux formes sont disjointes ou incluses)

Et on impose également les contraintes suivantes :

- *(R3) L'équivalence avec l'AdF scalaire.* Si $\mathcal{M} = \{\mathcal{S}_1\}$ alors $\mathcal{S}' = \mathcal{S}_1$ (si l'image est scalaire, alors la méthode doit produire le même arbre que l'AdF calculé sur l'image d'origine).

- *(R4)* Pour chaque forme $X \in \mathcal{S}$ telle que pour chaque autre forme $Y \in \mathcal{S}$, $X \cap Y = \emptyset$ ou $X \subseteq Y$ ou $Y \subseteq X$ alors, $X \in \mathcal{S}'$ (une forme qui ne chevauche aucune autre forme doit exister dans l'arbre final).

- *(R5) L'invariance par changement et inversion de contraste* Une transformation ψ est dite morphologique (i.e. invariante par changement de contraste) si pour une fonction croissante $g : \mathbb{R} \to \mathbb{R}$, $g(\psi(u)) = \psi(g(u))$. De plus, la transformation est dite auto-duale si elle est invariante par complémentation $\complement(u) = -u$ i.e. $\complement(\psi(u)) = \psi(\complement(u))$. Soit une fonction monotone $F : \mathbb{R} \to \mathbb{R}$ telle que $\forall x, y \in \mathbb{R}$, $x < y \Rightarrow F(x) < F(y)$ ou $F(x) > F(y)$, ψ est à la fois auto-duale et invariante par changement de contraste si $F(\psi(u)) = \psi(F(u))$. L'AdF est un support pour certains opérateurs morphologiques auto-duaux et une représentation T est dite auto-duale et morphologique si $T(F(u)) = T(u)$. Pour l'extension au multivarié de cette propriété, notons $\mathcal{F}$ l'ensemble de toutes les fonctions monotones, T est dit invariant par changement et inversion de contraste si pour $\mathbf{F} \in \mathcal{F}^n, \mathbf{F}(\mathbf{x}) = (F_1(x_1), F_2(x_2), \ldots, F_n(x_n))$, alors $T(\mathbf{F}(\mathbf{u})) = T(\mathbf{u})$.

3.2 *Description de la méthode*

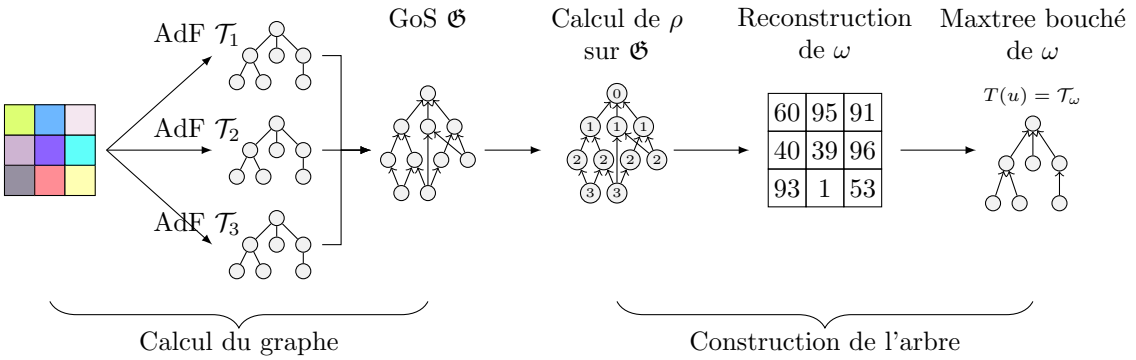


FIGURE 4: Les 5 étapes de la méthode proposée. (1) L'image d'entrée $\mathbf{u}$ est décomposée en ses différentes composantes $u_1, u_2, \ldots u_n$, (2) l'AdF est calculé sur chaque composante, (3) les AdFs sont fusionnés en une unique structure : le GdF, (4) une image scalaire ω est calculée en utilisant la profondeur des nœuds auxquels les points appartiennent. (5) un arbre est calculé à partir de ω.

La méthode que nous proposons est un processus en 5 étapes (voir la fig. 4). Tout d'abord, $\mathbf{u}$ est décomposé en ses composantes individuelles $u_1, u_2, \ldots, u_n$ sur lesquelles on calcule les AdFs $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ associés aux ensembles de formes $\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_n$. On note

$\mathcal{S} = \bigcup \mathcal{S}_i$, on appelle le GdF $\mathfrak{G}$ la couverture de $(S, \subseteq)$, i.e. c'est le graphe d'inclusion des toutes les formes issues de coupes marginales. Soit $\rho : \mathcal{P}(\Omega) \to \mathbb{N}$ un attribut de forme algébrique decroissant i.e. $\forall A, B \in \mathcal{S}, A \subset B \Rightarrow \rho(A) > \rho(B)$

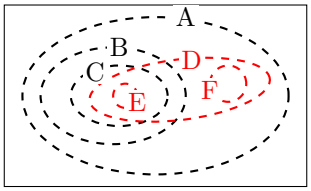
Un exemple d'un tel opérateur est l'attribut de *profondeur*. La profondeur d'une forme A de $\mathfrak{G}$ est la longueur du chemin le plus long de la racine vers A. ω Soit $\omega : \Omega \to \mathbb{R}$ défini par :

$$\omega(x) = \max_{X \in \mathcal{S}, x \in X} \rho(X) \tag{1}$$

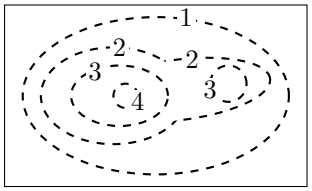
ω est une image scalaire qui associe en chaque point de x, la profondeur de la forme la plus profonde qui le contient (voir les figs. 5b and 5c) Soit $\mathbb{C} = \{CC([\omega \geq h]), h \in \mathbb{R}\}$. $(\mathbb{C}, \subseteq)$ est en fait le max-tree de ω. Ce dernier pouvant donner lieu à des composantes avec des trous, on considère donc $\mathcal{S}' = \mathcal{H}(\mathbb{C})$ et $(\mathcal{S}', \subseteq)$ pour l'AdF final $\mathcal{T}_\omega$ (voir la fig. 5d).

Intuition. On explique maintenant l'intuition de cette démarche. D'abord, nous commençons par calculer les AdFs marginaux de $\mathbf{u}$ qui nous donnent un ensemble de formes initial. Les différents arbres (ensemble) fournissent une représentation de l'image d'origine et $\mathbf{u}$ peut être reconstruite marginalement depuis ces arbres. Cependant, la manipulation simultanée des différents arbres est délicate et il manque une information capitale : comment les formes d'un arbre donné sont liées (au sens de l'inclusion) aux formes des autres arbres. Le graphe $\mathfrak{G}$ n'est rien de plus que ces arbres fusionnés en une unique structure et ajoute la relation d'inclusion qui manquait précédemment. Par conséquence, $\mathfrak{G}$ est plus « riche » que $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ parce que la transformation de $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ à $\mathfrak{G}$ est réversible ; $\mathfrak{G}$ est une représentation complète de u (u peut être reconstruite depuis $\mathfrak{G}$). De plus, $\mathfrak{G}$ est invariant par changement et inversion de contraste marginal de u car $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ le sont.

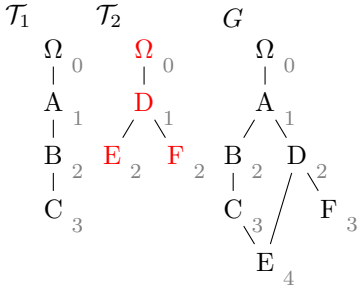
La seconde partie de la méthode tente d'extraire un arbre depuis $\mathfrak{G}$ vérifiant les contraintes données dans la section 3.1. Le problème majeur est d'obtenir un nouvel ensemble de formes depuis $\mathfrak{G}$ qui ne se chevauchent pas. Notons d'abord que pour n'importe quel attribut décroissant ρ, on a $(\mathcal{S}, \subset)$ qui est isomorphe à $(S, \mathcal{R})$ où $A \mathcal{R} B \Leftrightarrow \rho(A) > \rho(B)$ and $A \cap B \neq \emptyset$. En termes plus simples, la relation d'inclusion entre deux formes que nous voulons conserver peut s'exprimer en terme de valeurs d'attributs dans $\mathbb{R}$. Supposons maintenant que $(\mathcal{S}, \subset)$ est un arbre et considérons l'image $\omega(x) = \max_{x \in X, X \in \mathcal{S}} \rho(x)$, on a ainsi $\mathbb{C} = \{CC([\omega \geq h]), h \in \mathbb{R}\} = \mathcal{S}$. Dit autrement, le max-tree de l'image ω reconstruite à partir de l'attribut ρ évalué sur un arbre $\mathcal{T}$ donne exactement le même arbre $\mathcal{T}$ (pré-requis R3). Plus généralement, si une forme A ne chevauche aucune autre forme de $\mathcal{S}$ alors elle appartiendra à $CC([\omega \geq h])$ (pré-requis R4). Dans la section qui suit, nous allons maintenant justifier le choix de la profondeur comme attribut pour ρ.



(a) Image originale u (2-canaux) et ses formes.



(c) L'image ω construite depuis $\mathfrak{G}$



(b) Les AdFs marginaux $\mathcal{T}_1$, $\mathcal{T}_2$ et le GdF. La profondeur apparaît en gris clair près des nœuds.

(d) Le max-tree $\mathcal{T}_f$ of ω

FIGURE 5: Schéma illustrant le fonctionnement de la méthode

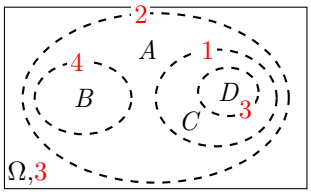
3.3 *Le calcul de la carte d'attribut ω*

La 4^{e} étape de la méthode consiste à choisir un attribut à évaluer sur le GdF $\mathfrak{G}$. C'est une étape critique qui décide de quelle forme va être fusionnée ou supprimée. Nous expliquons maintenant pourquoi nous utilisons la profondeur comme attribut pour fusionner les formes.

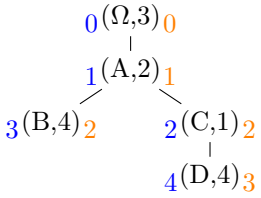
Considérons la distance entre deux points (p, p') dans Ω définie par

$$d_{\mathrm{TV}}(p, p') = \min_{C_{(p,p')}} \int_0^1 |\nabla u(C(t)).\dot{C}(t)|.dt \tag{2}$$

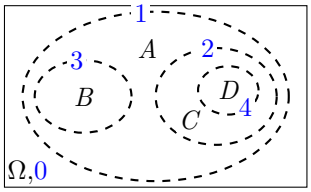
où $C(t)$ est un chemin dans Ω de p à p'. L'eq. (2) est en fait la variation totale (TV) du chemin minimisant la TV parmi tous les chemins de p vers p'. Cette mesure a par exemple été utilisée par DUBROVINA et al. [44] à des fins de segmentation où l'AdF est utilisé comme support pour le calcul efficace de la distance. Soit $\omega_{\mathrm{TV}}(x) = d_{\mathrm{TV}}(\partial\Omega, x)$, la carte de distance de variation totale depuis la bordure de l'image. Celle-ci peut être calculée par simple calcul d'attribut sur l'arbre : il suffit de sommer les variations absolues le long du chemin de la racine vers un nœud. Ainsi, au lieu de considérer l'arbre $\mathcal{T}$ des lignes de niveaux de u, on peut considérer à la place le max-tree $\mathcal{T}_\omega$ des lignes équidistantes de TV. Les deux arbres sont équivalents en niveaux de gris. Le problème avec la variation totale est qu'elle n'est pas invariante au changement de contraste de u. Une distance invariante



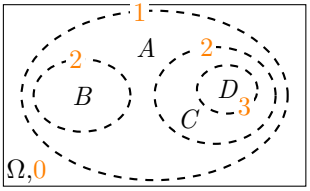
(a) Image originale u, ses formes et ses lignes de niveaux.



(b) L'AdF de u et l'évaluation de ρ_{TV} (bleu) et de ρ_{CV} (orange).



(c) Les lignes de niveaux de ω_{TV}



(d) Les lignes de niveaux de ω_{CV}

FIGURE 6: Équivalence entre les lignes de niveaux de l'image en niveaux de gris u et des lignes de niveaux des cartes de distances ω_{TV} et ω_{CV}.

par changement de contraste de u serait simplement de compter le nombre de variation, *i.e.*, le nombre minimum de ligne de niveaux à traverser pour atteindre p :

$$d_{\mathrm{CV}}(p,p') = \min_{C_{(p,p')}} \int_0^1 \mathbb{1}\{\nabla u(C(t)).\dot{C}(t)\}.dt \tag{3}$$

Algorithmiquement parlant, construire ω_{CV} consiste à calculer l'attribut de profondeur $\rho_{\mathrm{CV}}(A) = |\{X \in \mathcal{S} \mid A \subset X\}|$ et reconstruire $\omega_{\mathrm{CV}}(x) = \max_{X\in\mathcal{S}, x\in X} \rho_{\mathrm{CV}}(X)$ (voir la fig. 6).

Basé sur l'équivalence entre les lignes de niveaux et les lignes « d'équidistances » en niveaux de gris, on peut étendre cette idée en couleur. Comme dans l'eq. (3), on souhaite compter le nombre minimum de lignes de niveaux marginales à traverser. Plus formellement :

$$\rho(A) = \max_{\phi \in [\Omega \rightsquigarrow A)} |\phi| \text{ and } \omega_{\mathrm{CV}}(x) = \max_{X\in\mathcal{S}, x\in X} \rho(X)$$

où $[\Omega \rightsquigarrow A)$ désigne l'ensemble de tous les chemins de la racine vers A dans $\mathfrak{G}$. On compte ainsi le nombre de formes *qui s'incluent* à traverser pour atteindre la forme la plus profonde contenant x. ρ peut être calculé efficacement depuis $\mathfrak{G}$ en utilisant un algorithme classique de plus court chemin.

4 APPLICATIONS

4.1 *Filtres de grains*

Le filtre de grain [29, 108] est un opérateur qui suppriment les régions de l'image qui sont des extrema locaux dont l'aire est inférieure à un certain seuil. En ce sens, il est

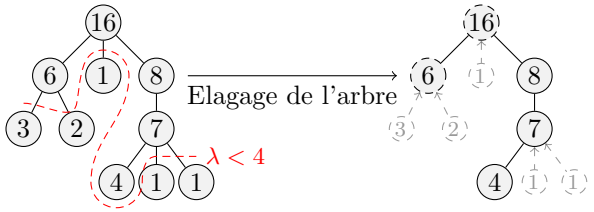
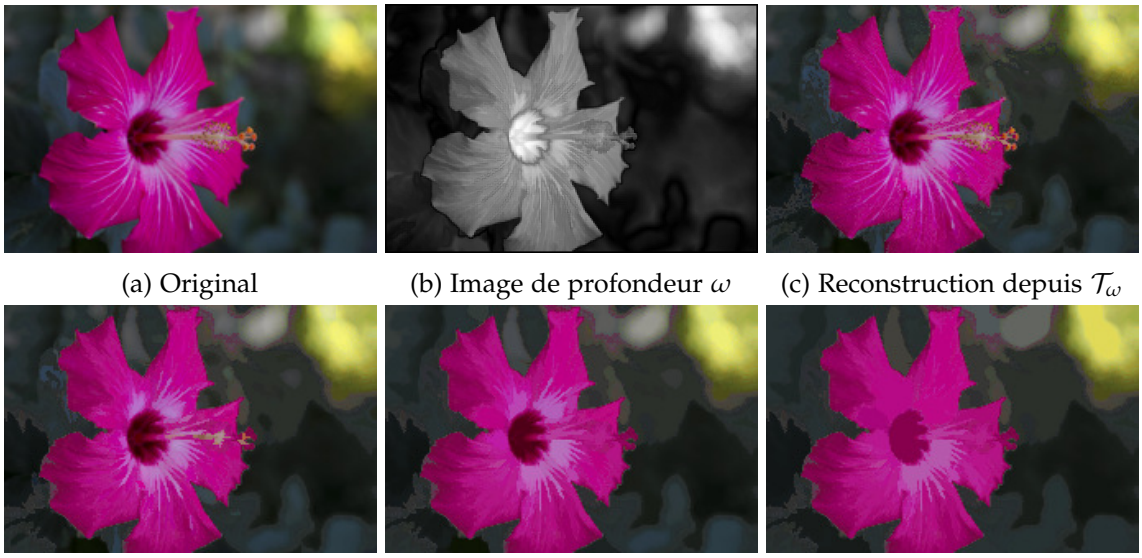


FIGURE 7: Schéma du processus de filtrage de grain.



(a) Original (b) Image de profondeur ω (c) Reconstruction depuis $\mathcal{T}_\omega$

(d) Filtres de grain 10, 100 et 500

FIGURE 8: Filtres de grains

lié aux *extreme filters* mais assure un traitement complétement symétrique des minima et maxima, *i.e.*, il est *auto-dual*. Avec l'AdF, un filtre de grain est un simple élagage de l'arbre, supprimant tous les nœuds qui ne passent pas un certain critère de taille. Le processus est illustré schématiquement dans la fig. 7. On assigne à chaque nœud la taille de la composante qu'il représente et on supprime les nœuds de taille inférieure à 4, *i.e.*, tous les nœuds en dessous de la courbe rouge. Les pixels qu'ils contiennent sont ensuite rattachés au plus proche ancêtre encore « vivants ». Malgré sa simplicité, nous montrons la puissance de ce filtre à travers la simplification et l'extraction de canevas de document.

Simplification d'images

Les filtres de grain permettent de révéler la « validité » de l'arbre dans le sens où une petite taille de grain doit supprimer ce que nous percevons comme bruit ou détail, alors qu'une grande taille de filtrage doit montrer les objets principaux et la structure de l'image. Dans la fig. 8, nous montrons la carte d'inclusion ω calculée par notre méthode et l'image reconstruite depuis le max-tree $\mathcal{T}_\omega$. La reconstruction consiste en le calcul, pour

chaque nœud, de la couleur moyenne de ses pixels, puis en l'affectation de cette valeur aux pixels. Puisque $\mathcal{T}_\omega$ n'est pas une représentation inversible de $\mathbf{u}$, cette dernière ne peut pas être reconstruite depuis $\mathcal{T}_\omega$. Néanmoins, la reconstruction est proche de l'image d'origine. Dans la fig. 8d, nous avons appliqué des filtres de grain de tailles croissantes qui suppriment les détails de manière « sensée » et fournit une reconstruction avec peu d'artefacts de couleur, ce qui valide l'organisation structurelle de notre arbre.

Extraction du canevas de document

FIGURE 9: Filtrages pour la extraction de canevas de document. Première ligne : images d'origine ; seconde ligne : résultats des filtres de grain.

Nous utilisons un filtre de grain pour extraire les boîtes de texte et les parties graphiques du document. En effet, les parties textuelles sont composées de lettres qui sont de petites composantes si l'AdFM est bien-formé. Au contraire, les boîtes de texte et le

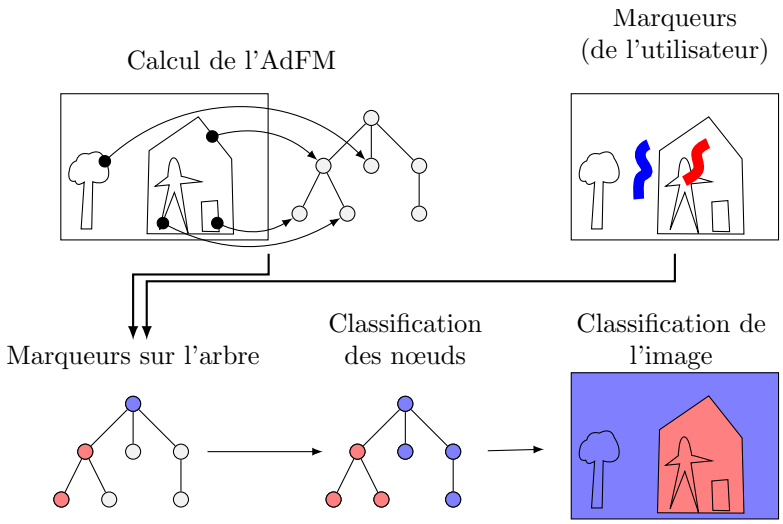


FIGURE 10: Schéma de la méthode de segmentation interactive.

contenu graphique sont de larges composantes qui persistent après le filtrage. Figure 9 montre l'extraction de contenu non-textuel où l'auto-dualité est fondamentale puisque le texte peut être au dessus d'un fond plus clair ou plus foncé. Comme chacun peut le voir, les images filtrées ne contiennent que les graphiques et les boîtes de texte alors que les lettres sont dans le résidu.

4.2 *Segmentation interactive*

Le problème de segmentation interactive peut être résumée comme suit. Étant donné deux ensembles disjoints de points F et B dans $\mathcal{P}(\Omega)$ représentant les marques rentrées par l'utilisateur, on souhaite annoter chaque point de Ω avec une des deux classes.

On considère la mesure entre deux points (p, p') de Ω :

$$d_{\mathrm{TV}}(p, p') = \min_{C_{pp'}} \int_0^1 |\nabla u(C_{pp'}(t)).\dot{C}_{pp'}(t)|.dt, \tag{4}$$

où $C_{pp'}(t)$ est un chemin dans Ω de p vers p', on peut calculer la distance entre un point x et les marqueurs F et B puis affecter à x l'étiquette de la classe la plus proche. Cette approche a été employée par [107, 12]. Néanmoins, dans [44], les auteurs ont montré que le calcul exact de cette distance nécessite une approche par ensemble de niveaux et requiert l'AdF.

Ainsi, l'idée fondamentale de la méthode est d'utiliser la représentation de l'image par l'AdFM de l'image au lieu de travailler directement sur son domaine d'origine. On applique ensuite le même principe que précédemment (une classification au plus proche voisin) mais en utilisant la topologie de l'arbre en lieu et place de la topologie de l'espace 2D. La segmentation finale est obtenue en reconstruisant l'image depuis l'arbre où tous

ses nœuds auront été annotés. La méthode peut donc se résumer aux étapes suivantes (voir aussi la fig. 10) :

1. Calculer l'AdFM $T(\mathbf{u})$ de l'image $\mathbf{u}$,
2. Valuer les arrêtes de $T(\mathbf{u})$ par la distance inter-nœuds (distance euclidienne entre les couleurs moyennes).
3. Étiqueter les nœuds de $T(\mathbf{u})$ depuis les marques entrées par l'utilisateur. On obtient ainsi deux ensembles de nœuds marqués pour $\mathcal{F}$ et $\mathcal{B}$,
4. Affecter tous les autres nœuds à l'une des classes $\mathcal{F}$ or $\mathcal{B}$ en calculant leur distance aux nœuds marqués (en utilisant la topologie de l'arbre), puis en assignant la classe du nœud le plus proche.
5. Reconstruire l'image depuis l'arbre annoté,
6. Nettoyer : conserver seulement les composantes connexes *objet* significatives (suppression des régions isolées de petite taille).

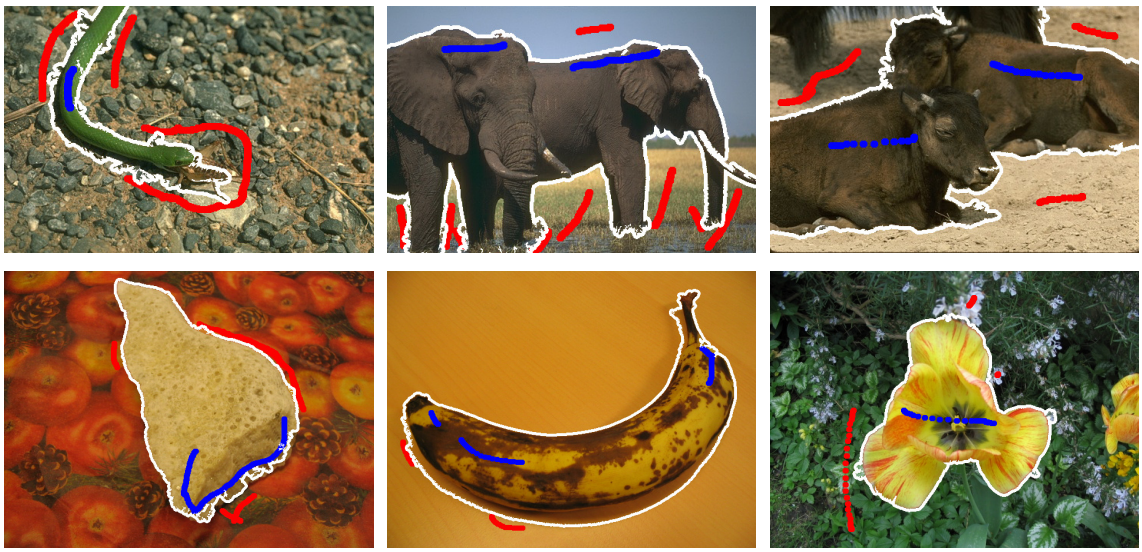


FIGURE 11: Segmentation intéractive avec notre méthode. Les marques rouges et bleues définissent respectivement le fond $\mathcal{B}$ et l'objet $\mathcal{F}$. La ligne blanche est la frontière $\mathcal{F}/\mathcal{B}$ calculée par notre méthode

Des exemples de résultat de notre méthode sont présentés dans la fig. 11.

Un avantage certain de travailler dans l'espace de formes est la capacité de récupérer de larges régions d'intérêts qui ne sont pas marqués par l'utilisateur. Cette propriété est intéressante pour les objets composés d'autres objets. Une forme étant une composante sans trous, il est suffisant de sélectionner la partie frontalière de l'objet englobant pour récupérer tous les objets.

Un second avantage de l'approche est qu'elle ne requiert aucune modélisation statistique de la région. Elle utilise uniquement les ensembles de niveau ce qui permet de récupérer de larges composantes avec peu de marques. La quantité de marques requises dépend en fait du nombre de lignes de niveau qui séparent le fond de l'objet.

4.3 *Détection de documents dans les vidéos*

Dans le cadre de la compétition ICDAR *Smartphone Document Capture and OCR (SmartDoc-2015) [19]*, on souhaite détecter automatiquement les documents dans des vidéos acquises par téléphones intelligents. La base de données couvre différents types de document (textuel et/ou avec un contenu graphique) et différents problèmes d'analyse de scène (changement d'illumination, bruit de déplacement, changement de perspective, etc.). Une nouvelle fois, la méthode que nous proposons s'appuie sur l'AdFM de l'image. Dit simplement, on cherche à identifier des nœuds dans l'arbre qui remplissent des critères de forme de document. Ceux-ci sont exprimés en tant qu'énergie à deux termes :

1. À quel point la bordure de la forme correspond à un quadrilatère. Pour chaque forme A, on calcule le meilleur quadrilatère Q_A correspondant à la forme et on mesure le ratio : $E_1(A) = |A|/|Q_A|$.

2. Quelle est la quantité de bruit dans l'objet. On s'attend à un document avec du texte qui se traduit par du bruit au niveau du document. Soit $\mathcal{L}_A = \{ X \in \mathcal{S} \mid X \subset A \text{ et } X \text{ est une feuille } \}$ l'ensemble des feuilles du sous-arbre enraciné en A alors :

$$E_2(A) = \frac{\sum_{X \in \mathcal{L}_A} (d(X) - d(A))}{|\mathcal{L}_A|}$$

   où $d(X)$ est la profondeur du nœud X

On cherche ensuite la forme qui minimise l'énergie $E_1(X) + E_2(X)$. Notons, que pour une meilleur précision de E_2, nous commençons par pré-traiter l'image avec un filtre de petit grain pour réduire l'effet du bruit naturel d'acquisition de l'image. Des exemples de résultats de notre méthode sont présentés dans la fig. 12.

Cette méthode (légèrement modifiée pour permettre le *tracking* de document entre les images vidéo) a obtenu la première place du concours. L'évaluation était basée sur le score Jaccard qui mesure la similarité entre l'ensemble des points attendus dans la vérité terrain et ceux retournés par la méthode. Celle-ci a obtenu un score moyen de 0.9716, variant entre 0.9710 et 0.9721 sur la base entière [19], ce qui montre la robustesse de l'approche proposée.

5 CONCLUSION

Nous avons présenté une méthode qui étend l'AdF sur les images multivariées. Contrairement aux approches standards, notre AdFM s'affranchit du choix d'un ordre total

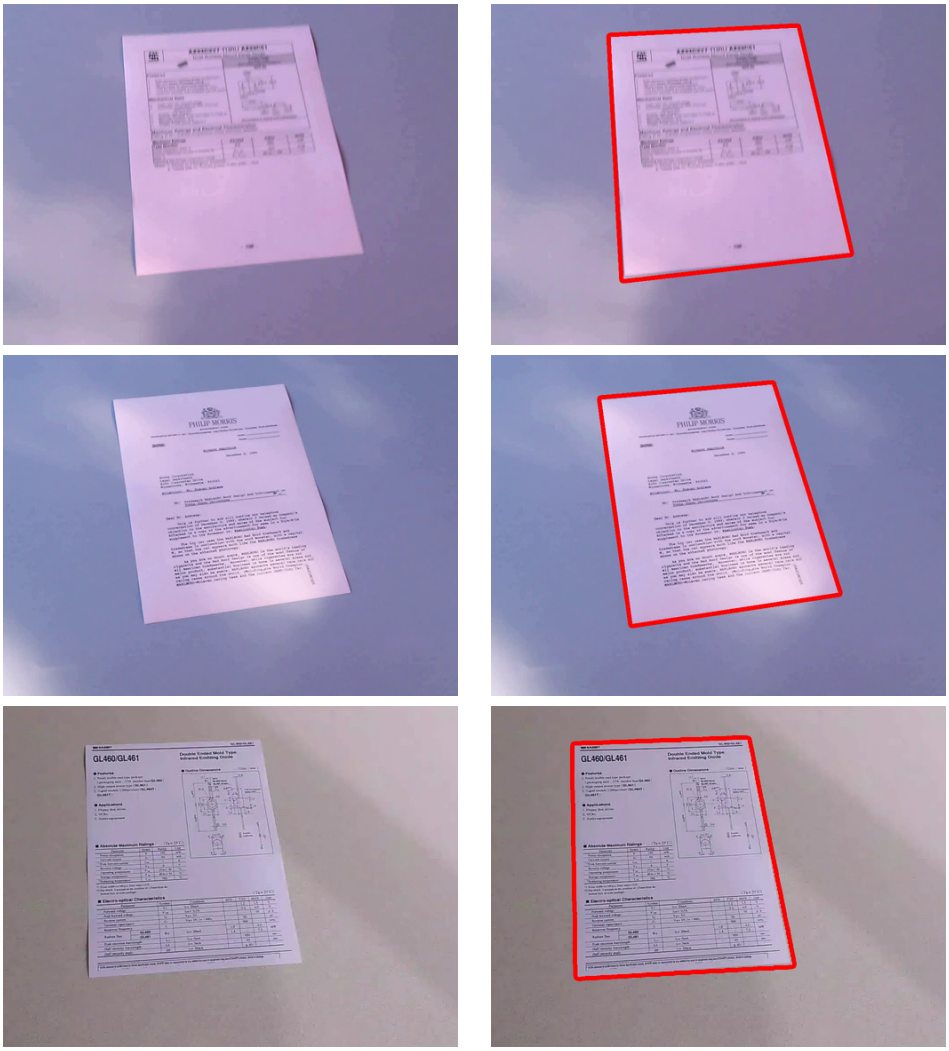


FIGURE 12: Compétition ICDAR sur la détection de document. Ces images montrent la robustesse de notre méthode au flou et aux effets spéculaires de lumière qui déplacent des frontières des objets. Des vidéos sont disponibles en tant que matériel supplémentaire [24].

sur ces données et repose uniquement sur la relation d'inclusion entre les formes. Elle produit ainsi un arbre qui est invariant par changement et inversion de contraste marginal de l'image. Nous avons tenté de mettre en évidence en quoi ces propriétés sont fondamentales en traitement d'images et en vision par ordinateur. Pour y parvenir, nous avons illustré AdFM à travers des applications de simplification, de segmentation et de détection d'objet, montrant des résultats déjà prometteurs et mettant ainsi en avant la versatilité et le potentiel de notre approche.

CONTENTS

1 INTRODUCTION

When *image processing* meets a higher level of understanding to become *pattern recognition* or *computer vision*, it is no longer sufficient to work at the pixel level. To retrieve some relevant information from images, *e.g.* for object detection, one needs to consider a local context (patch) or a region, generally at different scales. One may also need to know how these objects are related together, *e.g.* in scene understanding applications. Long story short, we require a representation of the image that renders its content and its organization. This one should enable:

- a multi-scale analysis of the image objects
- a spatial analysis of the image organization (in terms of objects inclusion or adjacency)

In this context, the Mathematical Morphology (MM) framework proposes tree-based image representations that fulfill previous requirements and feature also other interesting properties. They are typically of two kinds: hierarchies of segmentation and trees based on the threshold decomposition. They represent two different semantics. Hierarchies of segmentation render the adjacency of components, while component-trees describe how objects are included in each other. We are particularly interested in this second class because of their properties.

There are actually three kinds of trees based on the threshold decomposition: min-trees, max-trees, and Tree of Shapes (ToS). The min- and max-trees are the support for contrast invariant and morphological connected operators. In other words: **1**. they are based on the inclusion of connected components, so filtering these trees is a connected operation that does not move the object boundaries [112]; **2.** they are invariant by any contrast change. Yet, they are not invariant to illumination change but robust to *local* change of contrast as we expect the components to remain globally the same. The ToS [32, 28] (also known as *tree of level lines*) features the same properties as the min- and max-tree but it is also invariant by inversion of contrast. It can actually be seen as the "merge" of information hold by both min- and max-trees. This feature implies that: **3.** it is a self-dual representation of the image. This is fundamental in a context where structures may appear both on a brighter background or on a lighter one or if we cannot (do not want) make any assumption about the object/background layout. Self-dual operators have the ability to deal with both dark and light objects in a *symmetric* way [53, 117]; **4.** it allows a simple multiscale analysis of the image since the shapes are organized in a tree w.r.t. their inclusion; **5.** the level lines describe object boundaries in a *non-local* way. Contrary to many key-point detectors which rely on local information, level lines may be large Jordan close curves, orthogonal to the gradient, and fitting object contours [20].

Hierarchies of segmentation and threshold-decomposition-based trees are both widely used in the MM community and the choice of one representation *versus* another is application-dependant. In other words, our matter is not to say that a representation is better than the other (experts have to choose the right tool for their problems), but we aim at studying their ability to handle non-standard images. In particular, when the data to process is not a simple grayscale image.

With the wide variety of acquisition devices comes a wide range of different types of images to process. We focus on images where pixels are not scalar. In the following, we restrict ourselves on 2D images; yet everything presented in this report naturally applies to nD images. A well-known case of non-grayscale images is natural images acquired by cameras. Standard cameras output 24-bits 3-channel RGB images; one channel for each type of its red, green, and blue sensors. In computer graphics, it is also common to have an extra Alpha channel for transparency so that we are working on 4-channel RGB-A images. More recently, new devices such as Kinects allowed to render 3D spaces through a *Depth* component. Such devices produce 4-channel RGB-D images where the quantization of the *depth* channel is generally higher than for color channels. In medical imaging, the same object may be acquired with different type of scanners, *e.g.* PET and CT scans, yielding bi-modal or multi-modal images. To finish, another example can be found in the field of satellite imaging and remote sensing, where experts have to analyze images at multiple wavelengths of electromagnetic radiation (multispectral imaging). Some devices also produce images which narrow spectral bands over a continuous spectral range (hyperspectral and ultraspectral imaging). Through these examples, we see the need for hierarchical representations of multivariate images.

On the one hand, hierarchies of segmentation, which belong to the large class of hierarchical clustering methods, can easily be extended on multivariate data. Indeed, they rely on the notion of distances between pixels (or regions of pixel) in order to merge them incrementally into groups. There exist many distance functions between vectors that make sense, even if they actually depend of the image data type. For example, the ΔE distance, defined in the La*b* colorspace, is a rather "correct" measure between different colors as it is close to our "human" way of perceiving colors.

On the other hand, component-trees based on the threshold decomposition principle require a total ordering of values. Indeed, they rely on the inclusion of lower and upper level sets, which are nested only if the ordering is total.

As a consequence, many authors have been involved in defining a sensible total ordering on multivariate data. There exists a large amount of literature about the way of ordering colors (or more generally *vectors*). We distinguish two kind of approaches: the ones that try to get an ordering of the whole vector space and the others that only try to order the co-domain of the image (*i.e.*, only the set of values taken by the image). In the first group, the best-known total orderings are the lexicographical ordering, orderings based on the distance to a reference vector (or a reference sets of colors), or space-filling curves. Any method in this category is actually subject to the following problem: there will always exist two close points in the original high-dimensional space with a strong

discontinuity once projected on the 1D curve. That is why, some authors have investigated the second class of approaches: image-dependant total ordering. For example, Lezoray et al. [74] use manifold learning for dimensionality reduction in order to get best projection curve fitting the data; Chevallier and Angulo [35] use the same idea but with a clustering approach, etc. The problem with these approaches lies in the loss of some algebraic properties of the morphological operators built from them (*e.g.*, idempotence or duality). It motivates the fact we should not impose an "arbitrary" total order.

Passat and Naegel [102] were the first to investigate component-trees with a partial order. Actually, (except if the partial order is actually a lower or upper piece-wise total order), this partial order yields a much more complex structure which is no longer a tree but a graph. Apart from being expensive to compute, it also implies new filtering algorithms, new reconstructions strategies, and so on. In other words, it requires to adapt the whole component-tree framework (*i.e.*, segmentation, simplification, object detection methods...) to this new component-graph structure. It motivates the desire to manipulate a tree than a graph.

Problem statement

Once a total ordering is defined on values, the ToS can be computed, but we would lack some properties depending on the ordering used. Moreover, there is no actual consensus in the community about which total ordering makes the most"sense". On the other hand, with the same idea of Passat and Naegel [102], the component-graph could be extended to the ToS by considering every partial cut but we would then get a graph. In order to take part of the large literature about methods working on component-trees, we really prefer manipulating a tree instead of the graph.

The problem and the subject of this dissertation is how to get a *tree* for *multivariate* images which shares the properties with the ToS and *without imposing any "arbitrary" total ordering*?

Solution and contribution

The solution we propose is the Multivariate Tree of Shapes (MToS). Instead of defining a new total order, our method tries to merge some shapes computed marginally in the most "sensitive" way so that they finally form a tree. To that aim, an intermediate structure is introduced, the Graph of Shapes (GoS), from which the MToS is extracted using the graph topology only. The large context of our work is thus graph-based morphological representations of images, as reviewed in [93]. Our MToS features similar properties as the grayscale ToS. In particular, it is invariant to any marginal inversion or change of constrast of the image. Consequently, the contributions of this work are:

- a new hierarchical structure, the MToS, extending the ToS to multivariate images which features similar properties to the grayscale ToS, and, in particular, the invariance to any marginal inversion or change of constrast of the image;
- the proofs and the theoretical validity of our approach;
- efficient algorithms that enable to compute the MToS in quasi-linear time w.r.t. the number of pixels (and quadratics in the number of channels);
- the practical validity of our approach through applications covering different fields of image processing and computer vision, including: image filtering, object detection, image simplification, interactive and automatic segmentation, satellite image classification.

MANUSCRIPT ORGANIZATION

This dissertation is divided into three main parts. **Part i** introduces the main concepts of the work presented here: the hierarchical representations available with the MM framework and how they are related to multivariate data handling.

- **Chapter 2: Hierarchical Clustering Approaches.** This chapter is a quick review of the first type hierarchical representation offered by the MM framework, namely the *hierarchies of segmentation*. We discuss in this chapter the best-known structures, *i.e.*, the α-tree [118], the Binary Partition Tree (BPT) [111], and the hierarchical watershed [16]. We also highlight why they are interesting by describing some advanced manipulations available on those hierarchies, in particular, hierarchy re-weighting [52] and energy optimization on hierarchies [62].
- **Chapter 3: Trees Based on the Threshold Decomposition.** This chapter is a review of morphological hierarchies based on the threshold decomposition principle: the min- and max-trees [59] and the Tree of Shapes (ToS) [31], and highlights the difference in semantics with the hierarchies of segmentation. In this chapter, we also present the different strategies for filtering and reconstructing from these trees [114, 124].
- **Chapter 4: Morphological Trees Extended to Multivariate Images.** In this chapter, we review the classical approaches for extending the ToS on multivariate images. Since the problem lies in the lack of total orderings on the data, we review the "standard" approaches to impose a total ordering on vectors and their limitations. When the ordering relation lacks the anti-symmetry (*i.e.*, a preorder), the corresponding component trees and ToS may have nodes associated with different colors (vectors). This fact makes the reconstruction process more difficult as there is not a unique filtering value. As a consequence, we also review the different reconstruction strategies in the case of preorders.

- **Chapter 5: Component Graphs.** In this chapter, we review the new approach of Passat and Naegel [102] about the component-graph. They are the first to propose an extension of component-trees to partial orders. Their new structure, the *component-graph* can be easily transposed for the ToS as well. We also discuss in this chapter about the extra-complexity (both in terms of computational efficiency and ease-of-use) involved in manipulating a graph instead of a tree for the construction, the filtering and the reconstruction.

Part ii of this dissertation is the fundamental contribution of this work. It introduces our new approach to adapt the ToS on multivariate images.

- **Chapter 6: The Multivariate Tree of Shapes (MToS).** In this chapter, we introduce our new structure: the Multivariate Tree of Shapes (MToS). We present in this chapter an in-depth description of the method and the rational of the construction process.
- **Chapter 7: Properties of the MToS.** The MToS has properties similar to the gray-scale ToS, in particular, it is invariant to any marginal change or inversion of contrast of the image; which is what we consider as "self-duality" in the multivariate case. In this chapter, we prove that the method delivers a tree with the expected properties.
- **Chapter 8: MToS Computation Algorithm.** This chapter is dedicated to the algorithms involved in the computation of the ToS. We also provide a complexity analysis of the algorithms, as well as optimization tricks that makes the construction of the MToS in quasi-linear time w.r.t. the number of pixels and quadratics in the number of channels.
- **Chapter 9: Empirical validation of the MToS.** This chapter is an empirical validation of our structure. In particular, we compare our approach with other "standard" approaches extending the ToS on multivariate data in the case of image simplification. This chapter also provides some experiments about the validity of the structure in the presence of noise or when mixing channels with different dynamics, showing the robustness of the MToS to these "defaults".

In **part iii**, we show some applications using the MToS that validate practically the merits of our approach.

- **Chapter 10: Image filtering.** In this chapter, we propose to use the ToS for filtering purposes. We show how basic tree processing such as grain filters can be used for image simplification and document layout extraction. We have also applied the recent work of Xu et al. [144] about *shapings* that enables advanced type of filtering in a simple framework. This work has been applied in the context of filament filtering in bronchial cytology images.

- **Chapter 11: Image Segmentation and Classification.** In this chapter, we propose to use the MToS for image segmentation and classification. Three applications are depicted. The first one is an interactive image segmentation method where the user has to add some scribbles to the image to tag background and foreground areas, and the segmentation is done through tags propagation on the MToS. An automatic segmentation method is also proposed as an extension, as well as a way to get hierarchies of segmentation from the MToS. The second application is the document detection in videos in the scope of the ICDAR competition on Smartphone Document Capture and OCR. Our method, based on object filtering in the MToS obtained the first place of the challenge. The last application is the classification of hyperspectral images based on attributes profiles computed over the MToS.

In **Chapter 12: Conclusion and perspectives.**, we conclude this dissertation by a quick review of our novel approach extending the ToS on multivariate images, and the applications developed with it. We also present some possible improvements and future research on these applications. Eventually, we also give some suggestions for additional studies about algebraic aspects of the MToS.

Part I

A REVIEW OF CLASSICAL IMAGE HIERARCHIES

2 HIERARCHICAL CLUSTERING APPROACHES

The Mathematical Morphology (MM) framework offers two types of tree-based image representation: trees based on the threshold decomposition of the image and, hierarchies of segmentation also known as hierarchies of partition and pyramids [58]. The former will be discussed in chapter 3 while the later is the topic of this chapter. While hierarchies of segmentation are not new in image processing (the quadtree has been popularized in the 70's [57, 121]), they have recently got a regain of interest due to recent advances on energy optimization on hierarchies [52, 62, 116]. It has been shown to be useful for many applications such as simplification, segmentation [10, 22], image compression for thumbnail creation [111], etc.

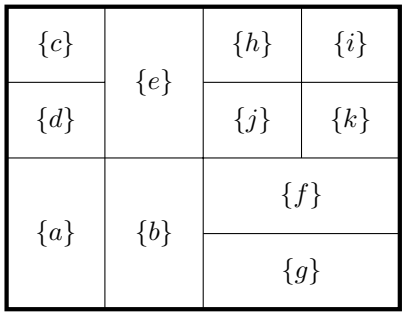
A partition π of an image $u : \Omega \to F$ is a set of components $\{ C_1, \ldots, C_n \}$ such that $\cup C_i = \Omega$ and any two components are disjoint. We say that π_i is a refinement of π_j (and we note $\pi_i \sqsubseteq \pi_j$) if $\forall A \in \pi_i$, $\exists B \in \pi_j$ such that $A \subseteq B$. Then, given the family $H = \{\pi_i, 0 \leq i \leq n \mid \pi_i \sqsubseteq \pi_{i+1}\}$, H forms a hierarchy. The finer partition π_0 is the leaves and $\pi_n = \{ \Omega \}$ is the root. Thus, the family H forms a chain of fine to coarse partitions. This is illustrated in fig. 13 where the hierarchy is represented by a dendrogram. In fact, the hierarchy provides a reduced search space for candidate regions of interest that might be useful for object spotting applications.

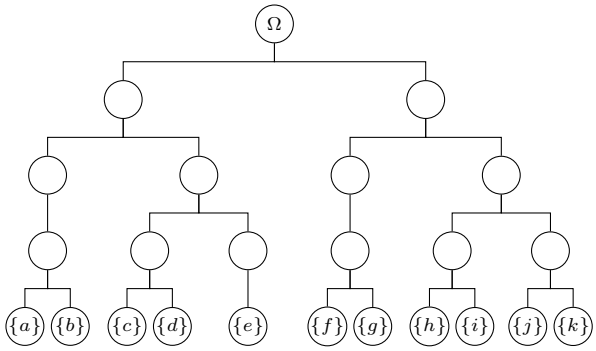
In this chapter, we will review in section 2.1 the classical hierarchies of segmentation used in MM. We will see that they share a same scheme; they are bottom-up clustering methods and that they have no troubles in dealing with multivariate data. In section 2.2, we will review some standard processings with those hierarchies.

2.1 WELL-KNOWN HIERARCHIES OF SEGMENTATIONS

In this chapter, we will consider the image in a graph framework. An image is a vertex/edge-weighted graph (V, E) where V, the set of vertices, stands for the pixels of the image and E, the set of edges, stands for the connection (4- or 8- connectivity). The weight of the vertex p is $u(p)$ (the original pixel value), and edges are weighted by a dissimilarity measure $\Delta(p, q)$ between two vertices p and q.

Because the clustering algorithms we will present only require a total ordering on the edges' weights and not on vertices', working with multivariate data is not a real issue. Any sensible distance between colors (or vectors) would fit perfectly (e.g. the CIE ΔE on the La*b* space).





The family H of partitions is:

$$
\begin{aligned}
\pi_0 &\quad \{\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g\},\{h\},\{i\},\{j\},\{k\}\} \\
\pi_1 &\quad \{\{a,b\},\{c,d\},\{e\},\{f,g\},\{h,i\},\{j,k\}\} \\
\pi_2 &\quad \{\{a,b\},\{c,d,e\},\{f,g\},\{h,i,j,k\}\} \\
\pi_3 &\quad \{\{a,b,c,d,e\},\{f,g,h,i,j,k\}\} \\
\pi_4 &\quad \{\Omega\}
\end{aligned}
$$

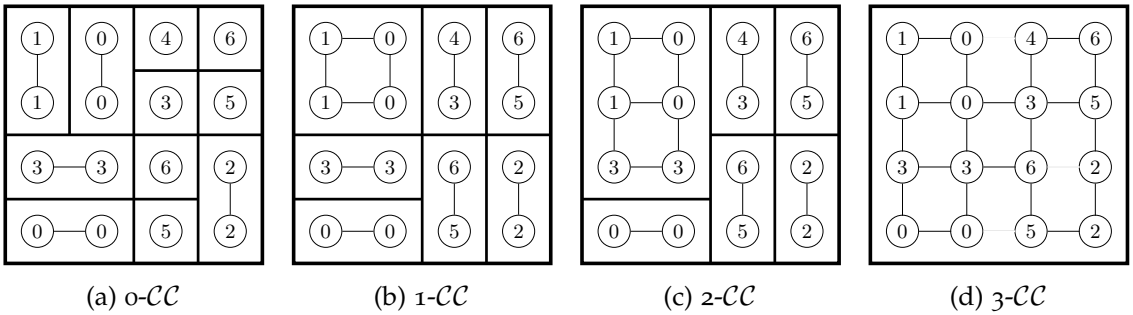
Figure 13: Example of a hierarchy of segmentation represented with a dendrogram. (Illustration from [62])

2.1.1 *α-tree and minimum spaning tree*

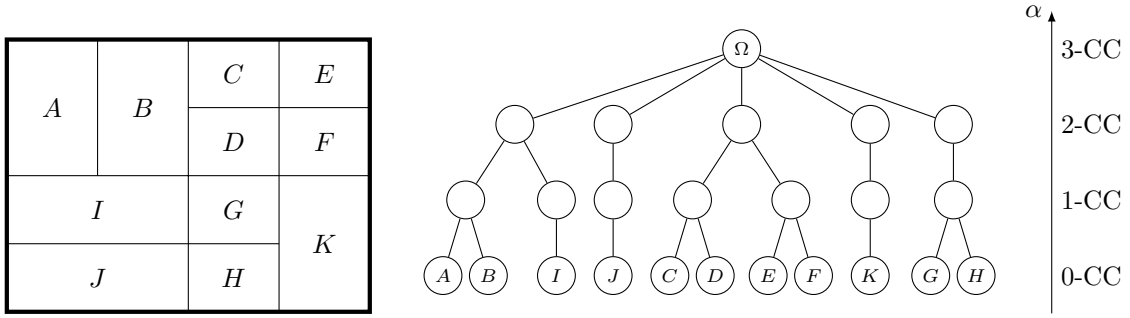
The α-tree [98], also known as quasi-flat zone hierarchy [84], relies on the notion of α-connectivity (or quasi-flat zone) to build a hierarchy [118]. We say that two vertices p and q are α-connected if there exists a path P from p to q such that $\max_{(p,q)\in P} \Delta(p,q) \leq \alpha$. It is straightforward that the α-connectivity defines an equivalence relation since it is reflexive, symmetric and transitive. We can then define α-connected components (α-$\mathcal{CC}$) which are the equivalence classes issued from the α-connectivity, *i.e.*, the maximal sets of points that are α-connected.

When α grows, two quasi-flat zones of lower α may merge to form a single region. As a consequence, the chain of partitions produced by the *alpha*-connected components forms a hierarchy. This is illustrated on fig. 14. The measure used to weigh edges is $\Delta(p,q) = |u(p) - u(q)|$. As one can see, there is a refinement from the 3-$\mathcal{CC}$ which is the whole image to the partition of 0-$\mathcal{CC}$ which is the partition of flat zones. The quasi-flat zones can thus be organized in a tree as shown in fig. 14e.

Note that when two regions A and B merge into AB, the distance from AB to another region C is the minimum of the distances of the subgroups to C (the edges other than the one of minimal weight are actually useless). This is why in the machine learning community, the α-tree is more commonly called single-linkage clustering and from an implementation point of view, it is closely related to the Kruskal's algorithm for



(a) 0-$\mathcal{CC}$ (b) 1-$\mathcal{CC}$ (c) 2-$\mathcal{CC}$ (d) 3-$\mathcal{CC}$



(e) Finest partition of flat zones (left) and the associated α-tree (right).

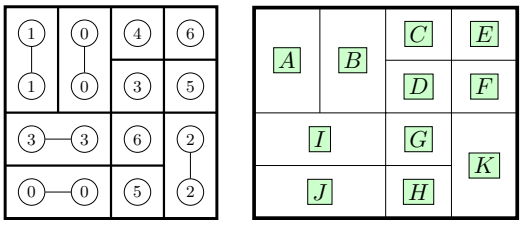
Figure 14: The hierarchy of quasi-flat zones illustrated on a 4×4 image.

minimum spanning trees which iteratively selects the edge of minimal weight to merge components [95].

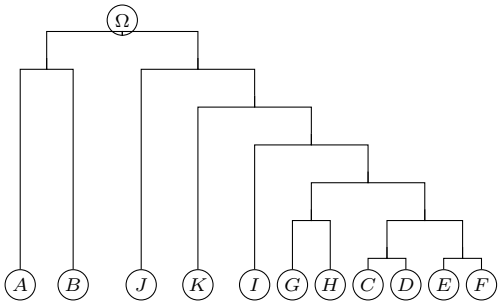
The hierarchy of quasi-flat zones is well-known to have two major problems. The first one is the chaining-effect in regions with low gradients (e.g., in blue sky) which makes the clusters grow fast and lead to heterogeneous regions even at the bottom of the tree. The second problem arises on noisy images. Because noise has a strong gradient with the surrounding region, it merges lately in the hierarchy and creates isolated, small regions in the segmentation. The same problem arises on region boundaries where the transitions are not sharp enough and edge pixels are isolated as well. For these reasons, a segmentation produced from the α-tree needs a bit of post-processing to remove those small regions.

2.1.2 *Binary Partition Tree (BPT)*

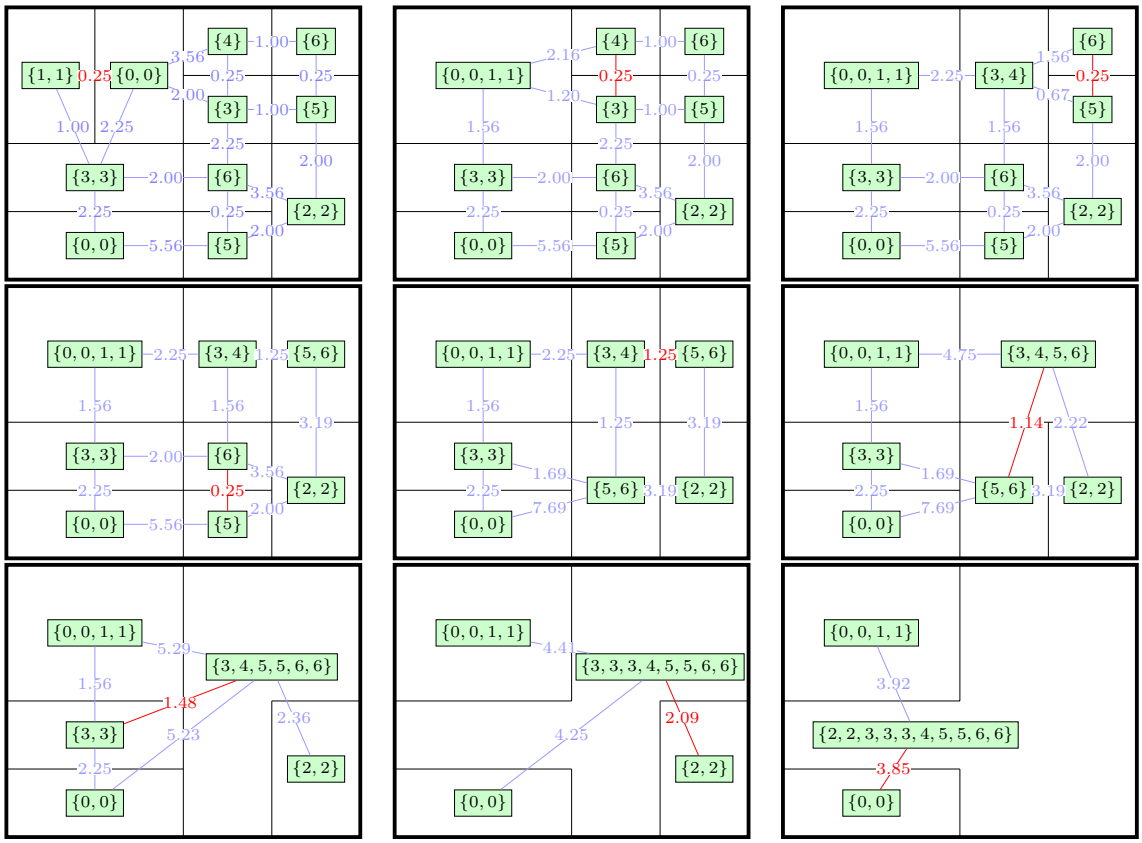
The α-tree presented in section 2.1.1 is actually a particular case of the BPT where the distance between two regions is the minimum of the gradient along the boundary of those regions. In the Binary Partition Tree (BPT) construction process, proposed by [111], one can introduce other merging criteria, for example, the homogeneity of two regions, or their shapes. The construction of the tree is thus driven by the merging criterion, and so by the final objective. Indeed, two important aspects of the method lie in the merging order and the region model, thus it depends on the application (see [129, 113]).



(a) Original image (left) and flat-zone labeling (right)



(b) BPT hierarchy.



(c) Step by step clustering. Red edges are the minimal weights indicating the regions that are going to be merged in the next step. The merging order is: A/B, C/D, E/F, G/H, CD/EF, CDEF/GH, CDEFGH/I, CDEFGHI/K, CDEFGHIK/J, CDEFGHIJK/AB.

Figure 15: The Binary Partition Tree (BPT) illustrated on an example.

Just like the α-tree, the BPT starts from an initial partition of the image (partition from the flat-zones or computed by a super-pixel algorithm). However, the α-tree does not require to update weights and distances during the *link* step of the algorithm, whereas the BPT generally does. The difference with the α-tree is illustrated on fig. 15. We use

the same original image as for fig. 14. We start from the initial partition given by the flat-zones and valuate edges with the variance as a cost to merge two regions. As for the Kruskal's minimum spanning tree algorithm, we iteratively select the edges of minimal weight, but once two regions merge, we have to update the value of the edges connecting with the neighbors by computing the new variance (see fig. 15c).

2.1.3 *Hierarchies of watershed*

The watershed transformation [17] is the best-known morphological method for image segmentation. The basic idea behind this method is to consider the image as a topographical surface and picturing water falling on it. Each drop of water will end in a catchment basin which are the local minima of the image. Each catchment basin thus has an influence area which attracts the drops. It defines a partition of the image in terms of those influence areas.

There exist many formulations of the problem as "watershed" denotes actually a class of algorithms. For example, Vincent and Soille [131] use an algorithm computing the geodesic influence zone of the catchment basins, Meyer [83] uses an immersion algorithm that expands the set of minima while keeping the same number of connected components as the number of minima. More recently the topological watershed [38, 39] has been introduced to overcome some issues related to the placement of the watershed lines separating two catchment basins. More precisely, one would like to have the pass value (the level when two basins merge) to be on the contour of the regions. As noted in [92], this property is important to compute some measure of contrast between the basins, which is at the basis of hierarchical watersheds.

Beucher [16] introduced the first hierarchy of segmentation based on watersheds. Given an initial partition of the image (given by a watershed transform algorithm), they successively simplify the partition by removing the mimima of the image. The order of removal of the minima actually implies the merging order of the catchment basins and defines a hierarchical clustering, so one needs to define a "persistence" measure of each minimum.

In [16], the authors use the dynamics [49] of the basin as the merging criterion. Basically, the dynamics of a minimum is the elevation distance that one must climb to reach another catchment basin, *i.e.*, this is the passing value minus the level of the current local minimum. In [92], other criteria are proposed such as the volume or the area (they have shown better performance in [105]), but any other attribute could be used. An example of hierarchies of segmentation based on the watershed is depicted in fig. 16 through saliency maps, *i.e.*, the region boundaries are weighed by their level of merging in the hierarchy. Just like with the dendrogram representation, any cut in the saliency map yields a partition of the image. Figure 16 also shows the importance of the criterion chosen to build the hierarchy. Indeed, the flooding by dynamics fails to correctly segment the ducks in the coarsest partition which splits the background into two parts. On the

contrary, flooding by area and volume give the ducks as the most salient objects, so they merge lately with the background in the hierarchy.

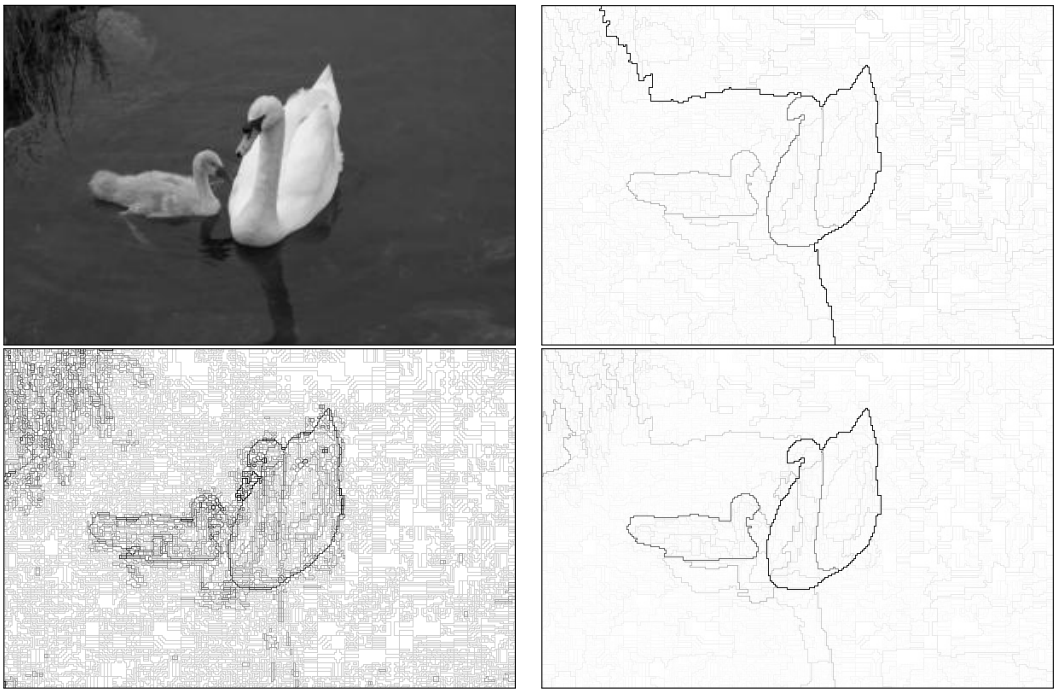
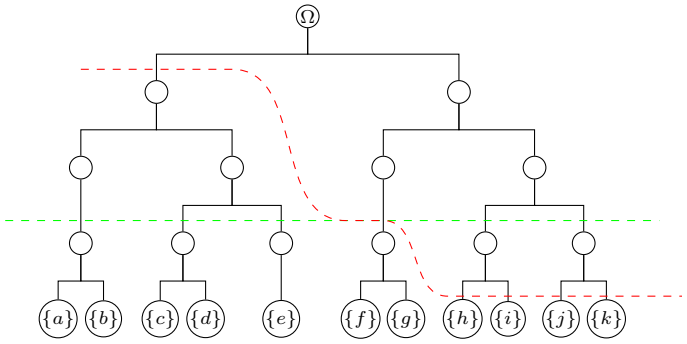


Figure 16: Hierarchies of watershed. Top left: original, top right: flooding by dynamics, bottom left: flooding by area, bottom right: flooding by volume. Hierarchies are represented through their saliency map.

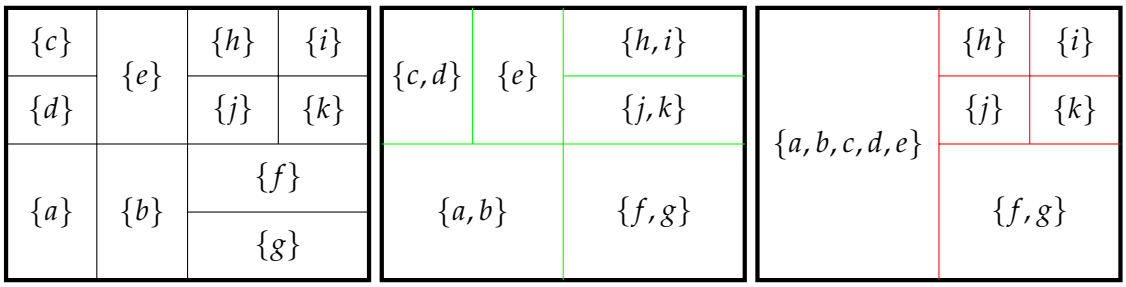
2.2 PROCESSING AND TRANSFORMING HIERARCHIES

2.2.1 *Horizontal and non-horizontal cuts*

Given a hierarchy $H = \{ \pi_0, \ldots, \pi_n \}$, any partition of $\pi = \{ C_1, \ldots, C_n \}$ defines a cut on the hierarchy, if every C_i is a class of H . If $\pi \in H$, the cut is said to be "horizontal" as it means cutting the hierarchy at a given level (or equivalently, thresholding the corresponding saliency map). On the other hand, the partition may be associated with several components from different π_i so that different levels of the hierarchy may be involved. This is illustrated on fig. 17. The horizontal green cut gives the partition π_2 which was originally in the family H (see fig. 13) while the non-horizontal red cut gives a new partition that was not in H and grows the size of the search space for possible segmentation.



(a) Hierarchy with an horizontal cut (green) and non-horizontal cut (red).



(b) Left: finest partition. Middle: partition from the green cut. Right: partition from the red cut.

Figure 17: Horizontal and non-horizontal cut on a hierarchy.

2.2.2 *Reweighing and energy optimization on hierarchies*

We have seen in section 2.1.3 that some merging criteria (flooding by area, volume, dynamics...) could affect the quality of the resulting hierarchy of watershed. This can be seen as a way of reweighing the hierarchy.

In section 2.1.1, we have introduced a hierarchical structure based on the notion of quasi-flat zones, namely the α-tree. In [118], the authors proposed the constrained-connectivity, as well as a way to modify the original hierarchy indexed by α to a new hierarchy indexed by the bounded variation ω of the component to limit the *linking* effect. More formally, the bounded variation BV is defined as:

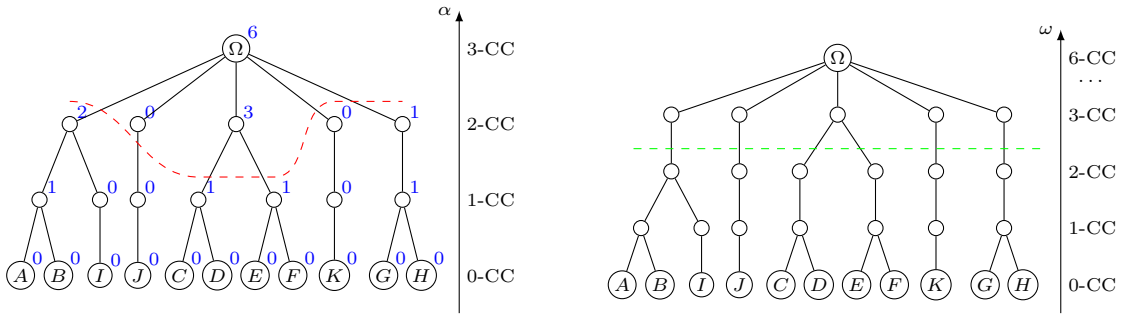
$$BV(\Gamma) = \max_{x \in \Gamma} u(x) - \min_{x \in \Gamma} u(x)$$

The ω-connected components do not form a partition of the image as they may overlap. Their idea is thus to limit themselves to α-connected components but constrained with the bounded variation ω. In other words, they are the largest α-connected components of bounded variation less than ω:

$$\omega\text{-}\mathcal{CC} = \text{Maximal elements of}_{\subseteq} \quad \{\Gamma,\ \Gamma \in \alpha\text{-}\mathcal{CC} \mid BV(\Gamma) \leq \omega\}$$

From now, any horizontal cut on the new ω-indexed hierarchy corresponds to a non-horizontal cut on the old α-indexed hierarchy. This is illustrated on fig. 18. The α-tree

given as an example in fig. 14 has been reweighed with the bounded variation. In other words, we compute the ω value of each region and reindex the hierarchy with these values. As one can see, it changes the topology of the tree and the horizontal green cut in the new hierarchy is related to a the non-horizontal red one in the original hierarchy.



(a) Hierarchy indexed by α. Blue labels are the bounded variation ω values of the regions.

(b) Hierarchy indexed by ω

Figure 18: α-tree hierarchy reweighed by ω. The horizontal green cut in (b) is equivalent to a non-horizontal cut in (a).

Energy optimization on hierarchies has been further studied in [51, 52] and later in [62]. Given a partition $\pi = \{ C_1, \ldots, C_n \}$ of the image, Guigues et al. [52] were interested in optimizing separable energies of the form:

$$E_\lambda(\pi) = \sum_{C_i \in \pi} C(C_i) + \lambda D(C_i) \tag{2.1}$$

where C is typically a fidelity term and D a term standing for the level of complexity of the model. The term C increases as the image simplifies while the term D decreases as the complexity of the model lowers. These classes of energy are well-known in image processing as they are used for simplification and segmentation (e.g. the Mumford-Shah energy [87], L_1 + Total Variation), or for image compression [111] (optimization on the rate/distortion trade-off). The way to solve this energy optimization and retrieve the best cut on hierarchies by dynamic programming has already been proposed in [111]. The novelty introduced by Guigues et al. [52] was to re-index the hierarchy in terms of the regularization parameter λ so that we do not have to choose a priori the simplification strength but rather choose it after the hierarchy has been re-indexed. Yet, the re-indexation is a single bottom-up traversal solving a dynamic programming formulation of the problem. For each node x (partial partition) of the hierarchy, we need to compute $\lambda^*(x)$ which is the value for which it becomes more valuable to merge the sub-regions of x into a single region. More formally, let $E_\lambda(x) = C(x) + \lambda D(x)$ denotes the energy of x, then:

$$\lambda^*(x) = \min\{ \lambda \mid E_\lambda(x) \leq \sum_{\text{child } y \text{ of } x} E_\lambda(y) \}$$

In their recent work, Kiran and Serra [62] wondered what kind of energies can be optimized this way on hierarchies. In particular, they focused on the composition law that

makes the framework valid. For example, in [52], the Mumford-Shah energy is additive as the energy of partition is the sum of the energies of its regions (see eq. (2.1)). On the contrary, in [118], the energy based on the bounded variation, is sup-generated:

$$BV(\pi) = \bigvee_{C_i \in \pi} BV(C_i)$$

The authors came out with climbing energies which must verify singularity and h-increasingness, *i.e.*, the energy of any node x must differ from the energy of any partition of x and for any two partial partitions π_1 and π_2 of x such that $\pi_1 \sqsubset \pi_2$, and a disjoint partial partition π_0:

$$E(\pi_1) \leq E(\pi_2) \Rightarrow E(\pi_1 \cup \pi_0) \leq E(\pi_2 \cup \pi_0)$$

This generalization to climbing energies and the study of new composition rules have enabled the introduction of new optimization constraints such as the number of classes in the simplification [61].

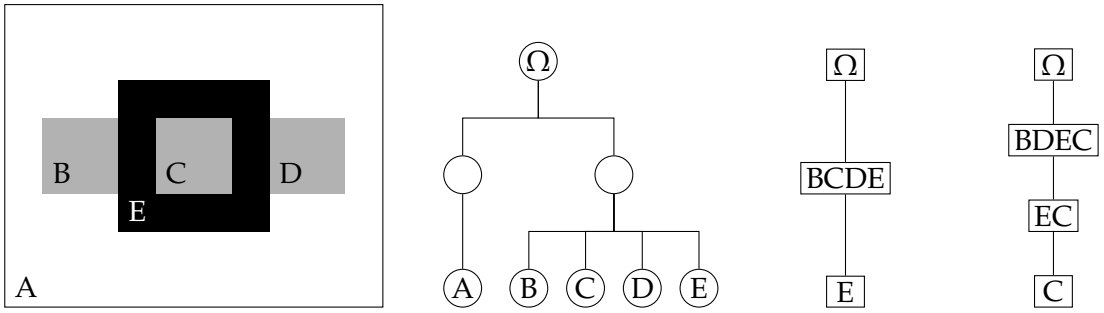
2.3 CONCLUSION

In this chapter, we have seen some tree representations of the image belonging of a wide class of hierarchies used in MM: the hierarchies of segmentation. The literature about this subject is quite large, and we have seen the keys ideas about the way to process, transform and optimize energies on those structures. From this chapter, there are two important points one should keep in mind before introducing the next chapter. First, these hierarchies of segmentation are based on clustering principles, so one basically has to define a distance or a cost function between pixels or regions. So there is no challenge about extending those hierarchies to multivariate images as sensible distances between vectors or colors already exist. The second point is that because they are based on clustering, the main information they encode is the adjacency and the level of disparity between adjacent regions. In the chapter 3, we will introduce another class of hierarchical representations based on the inclusion relation rather than the adjacency. We will see that their extension to multivariate images is not as straightforward.

3

TREES BASED ON THE THRESHOLD DECOMPOSITION

In this chapter, we review the three classical trees based on threshold decomposition of the image, namely, the min-tree, the max-tree and the Tree of Shapes (ToS). Min- and max- trees were introduced by Jones [59] and popularized by Salembier et al. [114] who found in them an efficient image representation adapted for shape recognition and image filtering. Max- and min-trees are dual, in the sense that the first one supposes that objects are lights regions over dark background and the other supposes the contrary. They will be reviewed in section 3.1, as well as the ways to process them. The ToS was introduced by [31] as a self-dual representation that merges min- and max-trees in a single structure. It is also known as *topographic map* as it encodes the level lines of the image and will be reviewed in section 3.2.



(a) Original (b) α-tree (c) Min-tree (d) ToS

Figure 19: The different semantic of the component trees vs. the hierarchy of segmentation.

These trees are fundamentally different from the ones we have exposed in chapter 2. First, any cut in a hierarchy of segmentation yields a partition; while in trees based on the threshold decomposition, a cut yields a partial partition. Second, hierarchies of segmentation renders the adjacency of regions while component trees and the ToS render the inclusion of objects. The different semantics are illustrated in fig. 19. Figure 19a represents the following scene (in a schematic way): "A gray belt with a black belt buckle over a white shirt". Figures 19b to 19d shows how the trees understand the scene.

- The α-tree does not see the belt as a whole but as three disjoint pieces of gray materials, separated by the buckle. They are adjacent to a white background.
- The min-tree sees the scene correctly. The gray belt contains of a black buckle and is over the white shirt.

- The ToS has one additional piece of information compared to the min-tree. It says that the buckle has one object inside, yet we do not know that this is the belt which shown through it.

3.1 MIN AND MAX-TREES

Component trees were introduced by Jones [59, 60] as efficient image representations that enable the computation of advanced connected filters in a simple way. Connected filters play an important role in Mathematical Morphology, because they preserve object boundaries. In the binary case, a connected filter preserves only the connected components that pass a given criterion, e.g., components that are large enough or that have a given shape. The same principle applies in gray-level through the notion of *threshold decomposition*. By thresholding the image with every level λ, we get a stack of binary image: the threshold sets. Binary connected operators are extended to gray-scale images by applying the filter on every threshold set and then recomposing the results. The component trees are actually hierarchical structures that encode the threshold sets and their inclusion relationship and allow efficient implementations of such gray-scale connected operators [114, 91, 15, 135, 133, 26].

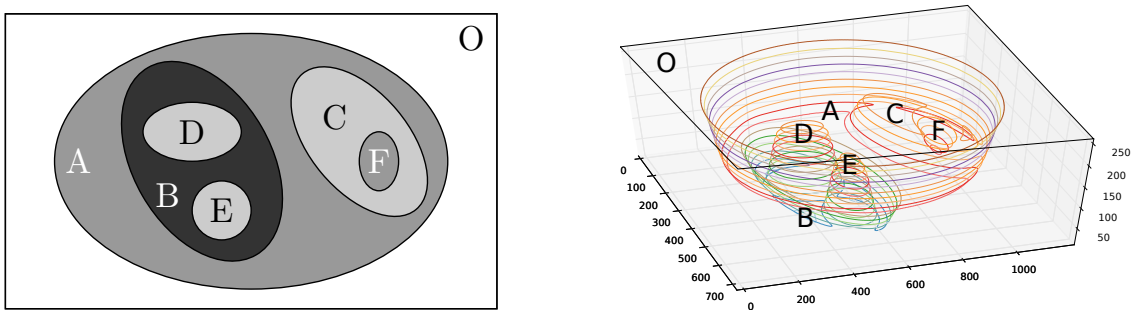
3.1.1 *Definition*

Let u an image from $\Omega \to F$ where F is endowed with an ordering relation $\leq$. We note the lower (resp. upper) threshold sets $[u \leq \lambda] = \{x,\, u(x) \leq \lambda\}$ (resp. $[u \geq \lambda] = \{x,\, u(x) \geq \lambda\}$). We note $\mathcal{CC}(X), X \in \mathcal{P}(\Omega)$ the set of connected components of X, $\Gamma^-_\lambda = \{\, X,\, X \in \mathcal{CC}([u \leq \lambda])\,\}$ and $\Gamma^+_\lambda = \{\, X,\, X \in \mathcal{CC}([u \geq \lambda])\,\}$ are the upper/lower peak components at level λ and finally, $\Gamma^- = \bigcup_\lambda \Gamma^-_\lambda$, $\Gamma^+ = \bigcup_\lambda \Gamma^+_\lambda$ denote the sets of lower and upper connected components. If the relation $\leq$ is total, any two connected components $X, Y \in \Gamma^-$ are either disjoint or nested. The set Γ^- endowed with the inclusion relation forms a tree called the *min-tree* and its dual tree, defined on Γ^+, is called the *max-tree*.

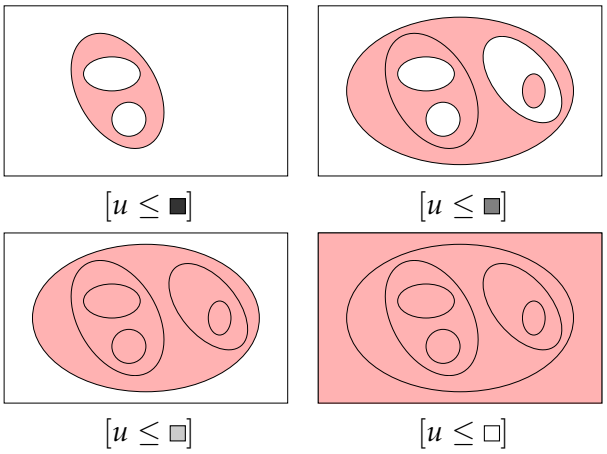
In a min-tree (resp. max-tree), a node represents a lower (resp. upper) connected component, *i.e.*, a set of points and the level of the threshold set. In fact, from the implementation point of view, the node $\mathcal{N}$ representing $\Gamma \in \Gamma^+_\lambda$ only stores the points at the level λ and Γ is actually the whole subtree rooted in $\mathcal{N}$ (see fig. 20).

3.1.2 *Filtering and reconstruction*

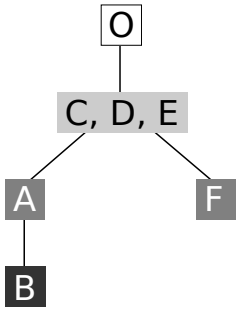
IMAGE RECONSTRUCTION Using the *decomposition principle*, the image u can be reconstructed from its threshold sets by the equations:



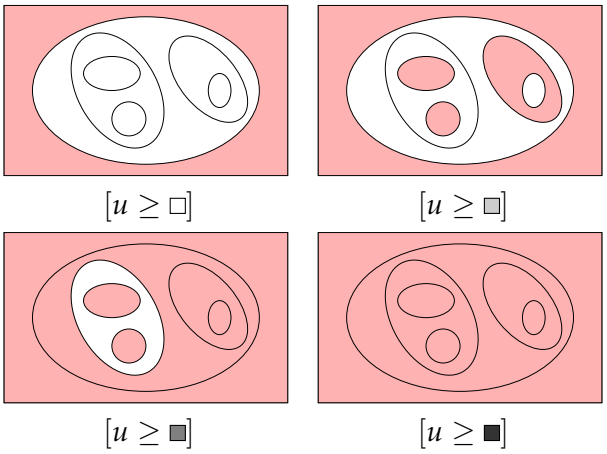
(a) Image and its 3d representation



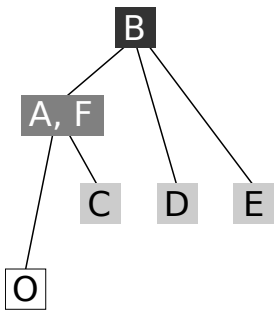
(b) Lower threshold sets.



(c) Min-Tree



(d) Upper threshold sets.



(e) Max-Tree

Figure 20: An image and its component trees.

$$u(x) = \bigwedge\{\lambda \in F \mid \lambda \geq u(x)\} = \bigwedge\{\lambda \in F \mid \Gamma \in \Gamma_\lambda^-, x \in \Gamma\}$$
$$u(x) = \bigvee\{\lambda \in F \mid \lambda \leq u(x)\} = \bigvee\{\lambda \in F \mid \Gamma \in \Gamma_\lambda^+, x \in \Gamma\} \tag{3.1}$$

Algorithmically speaking, the reconstruction using component trees is straightforward since each pixel gets valued with the level of the node it belongs to.

FILTERING THE COMPONENT-TREES Component trees provide a high-level representation of the image and describe the organization of the connected components in a hierarchical way. Thus, they enable us to perform advanced connected filtering in a simple way. For example, one can express the objects to be filtered out through an attribute (a criterion) that tells the components to be preserved and the ones to be removed. This attribute can be increasing (e.g., the size of the component, the dynamic...) or non-increasing (e.g., the perimeter, the inertia...).

In the first case, it leads to an attribute opening or closing (depending on the tree we are working with), in the second case it leads to an attribute thinning/thickening. More formally, let C be an increasing Boolean criterion. The trivial opening $\mathcal{C}(X) : \mathcal{P}(\Omega) \to \{\emptyset, X\}$ is just X if X meets C and the emptyset otherwise. Trivial openings allow the definition of attribute openings $\rho_{\mathcal{C}}(u)$ and attribute closings $\phi_{\mathcal{C}}(u)$:

$$\begin{aligned} \phi_{\mathcal{C}}(u)(x) &= \bigwedge \{\lambda \in F \mid x \in \mathcal{C}(\Gamma), \Gamma \in \Gamma_\lambda^-\} \\ \rho_{\mathcal{C}}(u)(x) &= \bigvee \{\lambda \in F \mid x \in \mathcal{C}(\Gamma), \Gamma \in \Gamma_\lambda^+\} \end{aligned} \tag{3.2}$$

It is straightforward that ρ and ϕ are increasing and idempotent because the criterion $\mathcal{C}$ is. One may prove that ϕ and ρ are respectively extensive and anti-extensive, which make them effectively openings and closings. In terms of tree processing, these operators are equivalent to pruning some branches. As soon as a node does not pass the criterion, the subtree rooted in that node is removed and its points are attached to the parent node.

If the criterion is non-increasing, eq. (3.2) leads to attribute thinnings and thickenings as ρ and ϕ are not increasing anymore. Salembier et al. [114] and later Urbach and Wilkinson [124] have defined specific filtering and restitution rules in the case of non-increasing attributes that can be categorized in two groups: *pruning* and *non-pruning* strategies. Pruning strategies involve preserving or removing some sub-branches of the tree, thus some nodes failing the criterion have to be kept while some others passing the test have to be discarded.

MIN A node is kept if it passes the criterion **and all** of its ancestors are kept.

MAX A node is kept if it passes the criterion **or any** of its descendants is kept.

VITERBI The selection of the nodes is based on an optimization problem where we have to compute the minimal path cost for each leaf (see [114]).

DIRECT A node is kept if it passes the criterion. The other nodes are discarded. When a node is discarded, the points are set to the level of the deepest *alive* ancestor, which is equivalent to the reconstruction formula given in eq. (3.2).

SUBTRACTIVE Same as the *direct* rule but the gray level difference Δ between a discarded node and its parent is passed to the descendants. This rule aims at preserving the same contrast between the remaining components in the residue image.

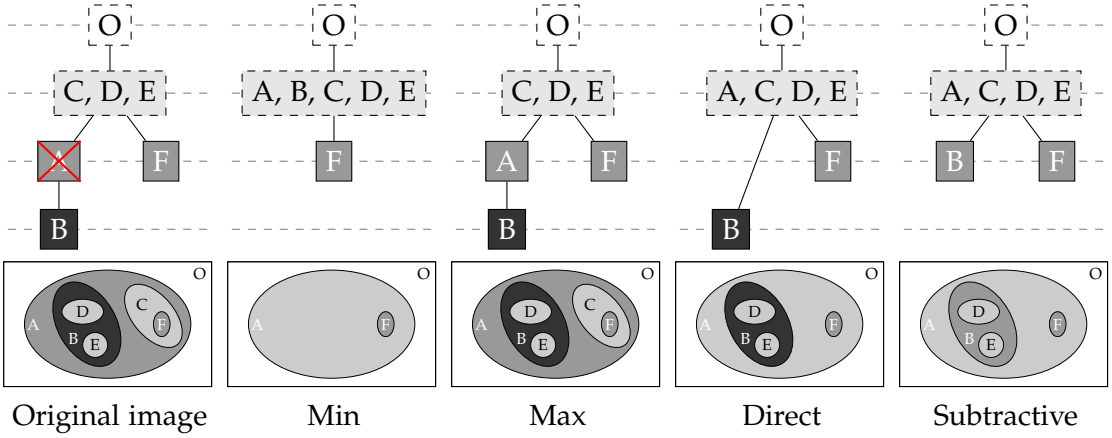


Figure 21: Illustrating the restitution process with pruning and non-pruning strategies.

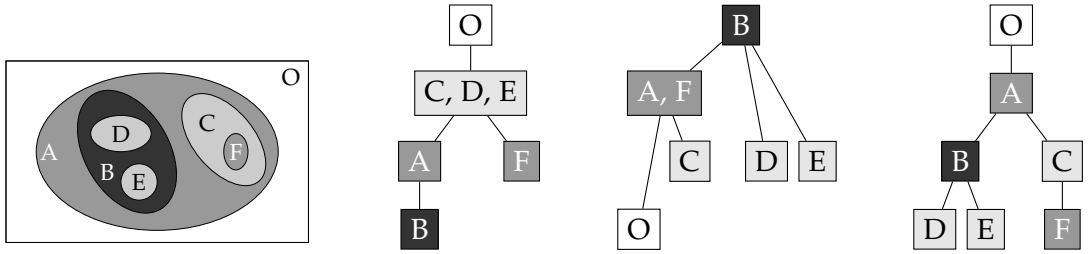
Figure 21 illustrates the differences between the min, max, direct and substractive strategies on the min-tree from fig. 20c. We used the criterion:
"$C(X)$ = number of holes in the component X ≤ 2" so that the node A is the only one that fails the criterion. Filtering with such a criterion is actually a "shape filter" in the sense of [124] since it is scale, rotation, and translation invariant and idempotent. Min and max rules are pruning strategies as they preserve or remove whole sub-trees as opposed to the *direct* and *subtractive* strategies which can remove nodes anywhere in the tree.

3.2 THE TREE OF SHAPES (TOS)

The Tree of Shapes (ToS), also known as *topographic map* is a hierarchical representation of a gray-level image in terms of the inclusion of its level lines [32, 85].

Every node of the ToS represents a connected component whose border is a level line and, typically, the number of nodes is close to the number of pixels for a non-degenerated image. The ToS of an image thus offers a description of the image contents as a collection of connected components, structured as a tree thanks to the inclusion of these components. Surprisingly this rich structure can be computed efficiently [46, 40], and can be also efficiently stored in memory [26].

It is linked to the min and max-trees introduced previously as it can be seen as the result of merging the pair of these dual component trees into a single tree. Since it is self-dual, it makes no assumption about the contrast of objects (either light object over dark background or the contrary). We only have one structure that represents the image contents so we do not have to juggle with the couple of dual trees. The ToS intrinsically eliminates the redundancy of information contained in those trees.



(a) Original image (b) Min-tree (c) Max-tree (d) ToS

Figure 22: The ToS and its differences with the Min and Max-Trees

3.2.1 *Definition and notation*

Let a set $X \subset \Omega$, we note ∂X the border of X and $\bar{X}$ the complementary of X. Let $\mathcal{H} : \mathcal{P}(\Omega) \to \mathcal{P}(\Omega)$ denote the hole-filling operator as defined:

$$\mathcal{H}(X) = \Omega \setminus CC(\bar{X}, \partial\Omega)$$

where $CC(\bar{X}, \partial\Omega)$ is the connected component of $\bar{X}$ connecting with the image border. Given that hole-filling operator, we call a *shape* any element of $\mathcal{S} = \{\mathcal{H}(\Gamma),\ \Gamma \in \Gamma_\lambda^-\}_\lambda \cup \{\mathcal{H}(\Gamma),\ \Gamma \in \Gamma_\lambda^+\}_\lambda$. If F is totally ordered, any two shapes are either disjoint or nested, hence the cover of $(\mathcal{S}, \subseteq)$ forms a tree called the *Tree of Shapes (ToS)* (in the following, as a matter of simplicity, we implicitly consider the cover of $(\mathcal{S}, \subseteq)$ while writing $(\mathcal{S}, \subseteq)$ only).

We see by this definition that the *shapes* are hole-filled connected components of the lower and upper threshold sets and so that the ToS can be seen as a "merge" of the Min and Max-trees. However, the hole-filling operation creates shapes that belong neither to the min-tree nor to the max-tree as shown in fig. 22.

Using the image representation in [46], one can ensure each level line is an isolevel closed curve given that the co-domain is totally ordered. Actually, the "totality" requirement comes from the definition of the level lines in terms of contours of lower or upper threshold sets and the level lines of u are actually the contours of the shapes. This way, the ToS encodes not only the inclusion of shapes but also the level lines inclusion.

Given $\mathcal{S}$ and a shape $A \in \mathcal{S}$, we will note $A\uparrow = \{X \in \mathcal{S},\ A \subseteq X\}$.

3.2.2 *Reconstruction from the ToS*

Caselles and Monasse [28, chap. 2.4] proposed two formulas for reconstructing the image from the ToS. The *direct* reconstruction which basically says that $\tilde{u}(x)$ is the level of the smallest shape containing x and the *indirect* reconstruction which first recovers the level sets from the shapes and uses eq. (3.1) to build the image. While the authors argue that the first formulation is not satisfactory as in the continuous case the smallest shape does not necessarily exist, it is sufficient in our case (discrete and with a finite number of shapes) and is closer to the algorithmic reconstruction process.

Consider the relation $\trianglelefteq$ on $\mathcal{P}(E) \times F$ defined as $(X_1, \lambda_1) \trianglelefteq (X_2, \lambda_2)$ iff $X_1 = X_2 \wedge \lambda_1 \leq \lambda_2$. Let G be the shape set $\mathcal{S}$ augmented by the level of the shape:

$$G = \underset{\trianglelefteq}{\text{Max. elements of}} \{(\mathcal{H}(\Gamma), \lambda),\ \Gamma \in \Gamma^-_\lambda\}_\lambda \bigcup \underset{\trianglelefteq}{\text{Min. elements of}} \{(\mathcal{H}(\Gamma), \lambda),\ \Gamma \in \Gamma^+_\lambda\}_\lambda$$

and the ordering $\preceq$ s.t. $(S_1, \lambda_1) \preceq (S_2, \lambda_2) \Leftrightarrow S_1 \subseteq S_2$

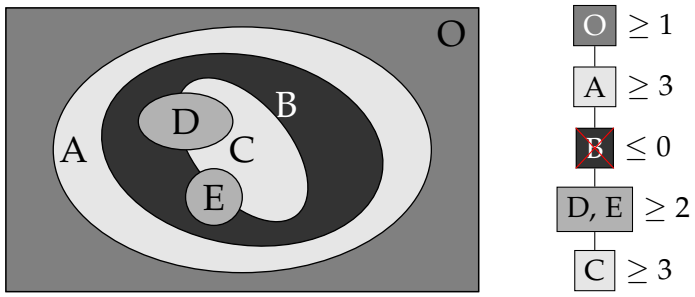
Then, the image can be reconstructed using:

$$\tilde{u}(x) = \lambda : (X, \lambda) = \bigwedge_{(S,\lambda) \in G, x \in S}^{\preceq} (S, \lambda) \tag{3.3}$$

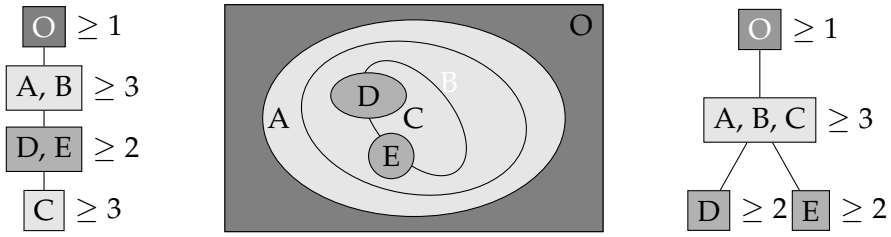
Again, as for the Min and Max-trees, the reconstruction using the ToS is straightforward since each pixel is stored in the smallest component it belongs to and so, gets valued with the level of its node.

3.2.3 *Filtering the ToS*

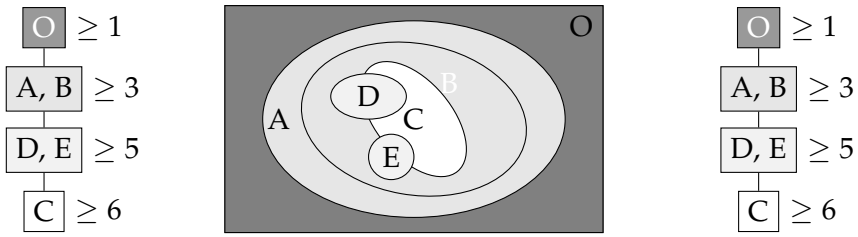
The same filtering strategies exposed in section 3.1.2 can be applied on the ToS. However, in the case of non-increasing criteria, the *direct* rule can lead to a reconstructed image that does not correspond to the tree from which the reconstruction was based. As a workaround, Caselles and Monasse [28, chap. 2.4] suggest to prevent the suppression of extremal shapes in monotone section of the tree if they have a child node of different type. Another strategy might be to use the *subtractive* strategy proposed by [124] for component trees as it preserves the contrast between the remaining level lines. This is illustrated in fig. 23. When removing the shape B by the *direct* rule, it creates a connection between A and C at the same level and changes the tree's topology. On the other hand, the *subtractive* strategy increases the level of D, E and F by the same amount of B and the topology is preserved.



(a) Original image and its ToS where we want to remove the shape B.



(b) Filtering with the *direct* rule. Left: filtered tree; middle: reconstructed image $\tilde{u}$; right: ToS of $\tilde{u}$.



(c) Filtering with the *subtractive* rule. Left: filtered tree; middle: reconstructed image $\tilde{u}$; right: ToS of $\tilde{u}$.

Figure 23: Non-pruning strategies illustrated on the ToS. The *direct* rule leads to a reconstructed image that differs from its original ToS.

4

MORPHOLOGICAL TREES EXTENDED TO MULTIVARIATE IMAGES

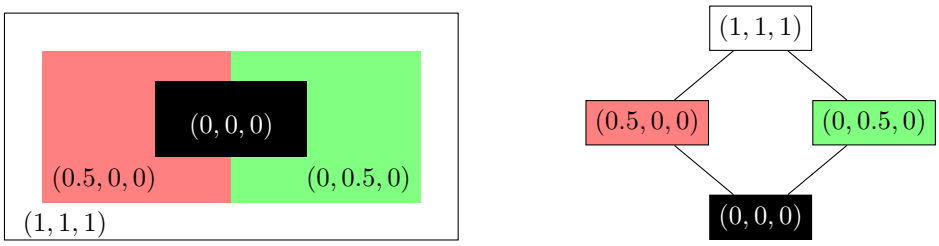
Mathematical morphology (MM) offers a non-linear image processing framework which is both simple and efficient. It has been widely used in many image processing tasks as for filtering, object detection, segmentation... Behind the scenes, MM operators rely on an ordering relation on values which must form a lattice. Those operators usually apply to binary and grayscale images, even if there are many attempts to deal with color images [8].

We have seen in chapter 3 a particular class of MM operators, named *connected operators*, that have been widely investigated thanks to their contour-preserving properties. These operators can be efficiently computed using component trees: min-tree, max-tree and ToS.

While most MM filters rely on a partial ordering relation only, component-trees and the ToS need the ordering to be total. The latter requirement hinders the extension of these structures to color images. In this chapter, we review the most important strategies to extend morphological tree processings to multivariate data, highlighting their problems w.r.t. self-duality and basics requirements of morphological filters.

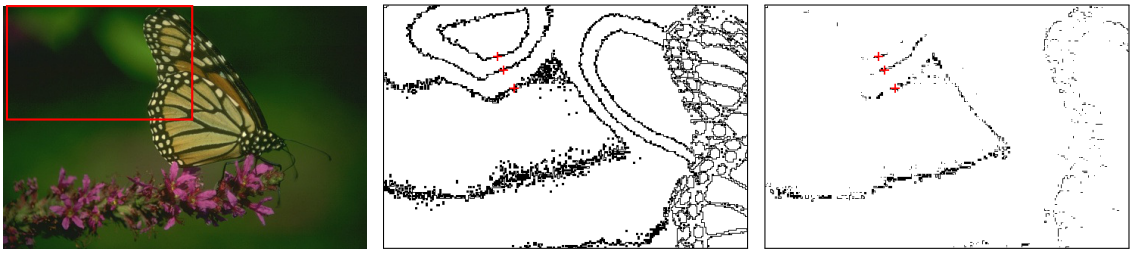
4.1 PROBLEM STATEMENT

As seen in chapter 3, the ToS (and component trees in general) relies on an ordering relation $\leq$ that needs to be total. On multivariate data, no natural total order exists, the "natural" product order is partial. As a consequence, the shapes from lower and upper sets may overlap without being nested. This is illustrated in fig. 24. Here, the cuts $[u \leq (0.5, 0, 0)]$ and $[u \leq (0, 0.5, 0)]$ intersect (via $[u \leq (0, 0, 0)]$) without being nested.



Original image. Hass diagram of the shape inclusion.

Figure 24: Problem with the ToS on a partial order. It leads to shapes that overlap without being nested.



(a) Original image (b) Iso-levels at three points on a gray level version of (a). (c) Iso-colors at the three same points on (a).

Figure 25: Level lines problem on $\mathbb{R}^n$.

Another way to see the problem, from a level lines standpoint, is that on $\mathbb{R}^n$ (n>1), level lines do not form closed curves as shown in fig. 25. We have selected three random points a, b, c and drawn every pixels $\{ x \in \Omega \mid u^*(x) = u(a)$ or $u^*(x) = u(b)$ or $u^*(x) = u(c) \}$ where u^* is the original image interpolated with intervals. On gray level (fig. 25b), the points form closed curves that do not cross and can be organized into a tree driven by the inclusion order of their interior. On the other hand, on colors (fig. 25c), the points do not form closed curves; the inclusion tree is degenerated with a low depth.

4.2 MULTIVARIATE MAGE PROCESSING STRATEGIES

4.2.1 *Marginal processing*

With a marginal approach, each component of the image is processed independently from the others. Then, the resulting images are merged back to form a multivariate image. Despite its simplicity, its major drawback is that it does not take into account the correlation between the different components and so a part of the information is discarded during the processing.

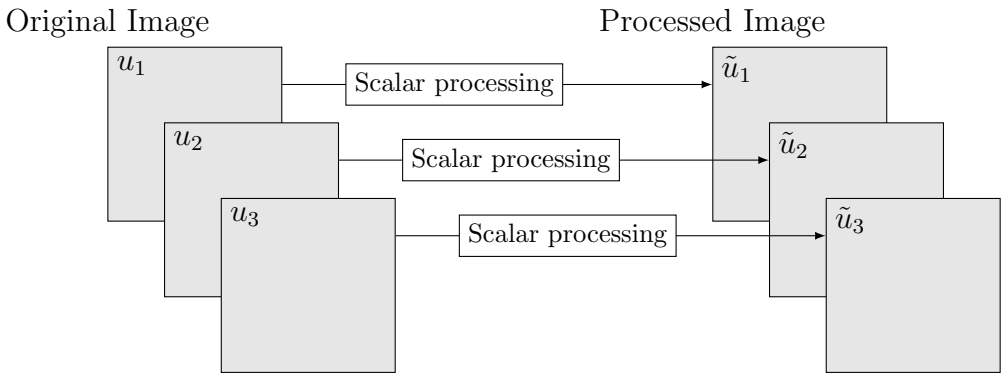


Figure 26: Marginal processing strategy.

Figure 27: False color problem of the marginal filtering with the ToS. (Grain size: 500)

In our case, this means computing the Tree of Shapes (ToS) of each image channel and processing each tree. Two majors problems emerge.

First, we do not have a single tree, but as many trees as the number of channels. As a consequence, we have to handle several trees at the same time. For some applications, such as object detection, this is a significant drawback since we may detect the same object many times and extra work is then required to remove these redundancies.

Second, in the case of filtering, the marginal approach leads to the well-known "false color" problem. It arises when recomposing the channels of the final image since it may create new values that were present in the original image. The presence of "false colors" may or may not present a problem, depending on the application. For example, for denoising, marginal processing offers a wider range of possible values in the output image and the new colors may be close to the original ones and do not alter visually the result [37]. On the other hand, for other application such as simplification or segmentation, it tends to create visible artifacts on the object boundaries due to the fact that the regions do not match exactly in the trees. This problem is illustrated on fig. 27 where a grain filter of size 500 has been applied marginally on each component. In the wall, the topologies of the three trees differ, the shapes do not merge at the same level of filtering. The recomposition process of the channels creates visible color artifacts.

4.2.2 *Vectorial processing*

Contrary to the marginal processing, a vectorial processing considers the set of the components as a whole. Depending on the application, this approach is favored because it prevents the creation of false colors. In our case, a vectorial processing would present a major advantage: we would have a single structure to process. However, it requires imposing a total ordering of vectors as required by the trees based on the threshold

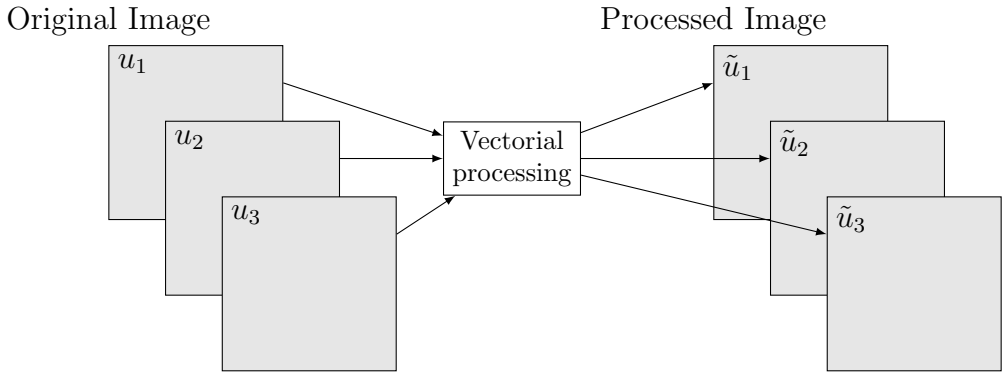


Figure 28: Vectorial processing strategy.

decomposition and the choice of particular total order is not straightforward as we will see in the next section.

4.3 ON THE IMPOSITION OF A TOTAL ORDERING

Among hierarchical representations of images, component trees rely on a data (pixel values) ordering that needs to be total. When dealing with multivariate data, authors generally have two approaches. They either define a new total ordering relation on their data, or adapt their structures to deal with the natural partial order. Many authors have attempted to define orders on colors. There actually exist more than seventy ways of ordering a vector space (see Aptoula and Lefèvre [9] for a rather complete survey of color orders used in the mathematical morphology framework). In this section, we review only the most well-known approaches to order vectorial data.

Let us first recall the definition of an ordering relation $\leq$. $\leq$ is a binary relation that is, $\forall x, y, z \in E$,

reflexive	$x \leq x$
transitive	$x \leq y$ and $y \leq z \Rightarrow x \leq z$
anti-symmetric	$x \leq y$ and $y \leq x \Rightarrow x = y$

Moreover, $\leq$ is said to be *total* if $\forall\, x, y$, $x \leq y$ or (not exclusive) $y \leq x$; if not, the relation is said to be *partial* because some elements are not comparable.

If $\leq$ does not meet the anti-symmetry property, then the ordering relation is a *preorder* (also known as a *weak* order if the relation is strict). The consequence is that two different values a and b may be considered equivalent ($a \leq b \wedge b \leq a$) by the relation. It leads to component trees where a node in the trees has several values (the ones belonging to the same equivalence class).

A classification of ordering relations can rely on their algebraic properties (totality, anti-symmetry...) or with respect to the way these relations are built. Barnett [14] proposed to classify them into four groups: marginal (M-ordering), conditional (C-ordering), partial

(P-ordering) and reduced (R-ordering). The former is a component-wise ordering that deals separately with each channel, *i.e.*, a partial order, and will be discussed in section 5.1. The three others provide total (pre)orders and are discussed below.

4.3.1 *Conditional orderings (C-ordering)*

With C-ordering, vectors are ordered by means of one, several, or all of their marginal components. The most well-known C-ordering is the lexicographical ordering, that is a total order. If only several components participate in the comparison, it yields a total preorder. For example let $v, w \in \mathbb{R}^n$, the lexicographical ordering $\leq_L$ using only the two first components ($v \leq_L w$ iff $v_1 < w_1 \vee v_1 = w_1 \wedge v_2 \leq w_2$) is a total preorder. Colors $(1, 1, 2)$ and $(1, 1, 3)$ are considered as *equivalent* by the above relation. The main pitfall of C-orderings lies in the importance given to the first components. Considering the RGB space for example, it implies that the red component is more relevant than the others. Workarounds like the sub-quantization of first components (also known as α-trimmed lexicographical ordering [97, 7, 115]) enables to lower the importance given to the first dimension.

4.3.2 *Reduced orderings (R-orderings)*

With R-ordering, vectors are reduced to scalar values using a mapping $h : \mathbb{R}^n \to \mathbb{R}$ s.t.

$$v < w \Leftrightarrow h(v) < h(w)$$

If h is injective then each index is mapped to a unique color and the relation is a total order, otherwise it is a total preorder.

In [127], R-orderings are classified in two main groups: R-orderings based on projections and the ones based on distances. The first group has mappings written as:

$$h(v) = \sum_{i=1}^{n} \lambda^i . v_i \tag{4.1}$$

The methods of this group employ standard unsupervised reduction techniques like the Principal Component Analysis.

The second group of methods, namely distance-based ordering, are supervised and they involve choosing a reference vector v_{ref} and the order relation is built upon a distance to v_{ref}, *i.e.*, $h(v) = d(v, v_{ref})$. More generally, a set of K reference vectors v^k_{ref} can be chosen and the mapping h has the generic form:

$$h(v) = \sum_{k=1}^{K} \lambda^k . d(v, v^k_{ref}) \tag{4.2}$$

The choice of a good distance function between vectors is another problem, e.g. authors generally use the CIELAB ΔE for colors. More advanced distances functions can be used,

specially those that depend on the image content. In [127, 128], the distance is built upon a support vector machine with a Gaussian kernel. Some reference points are chosen as background or foreground, they are the training set of the classifier. Then, a SVM regression is used to compute the distance to the other vectors, from which an ordering relation is deduced.

This type of ordering has been widely used[64], and studied in-depth in [6] to extend morphological filters. The first problem highlighted by [9] is the instability of the maximum (while the minimum is stable, close to v_{ref}) that makes the dilation unstable to slight modifications of the input image. This is a significant drawback from the perspective of designing self-dual filters. While erosion tends to the reference color v_{ref}, dilation tends to move away from the reference, to a "foreground" vector, but without defining it.

The second problem pointed out by Angulo [6] is the non-duality of the erosion and the dilation made from distance-based ordering (even in gray-level). This is illustrated in fig. 29. Values are in the range $[0.0 - 1.0]$ and $v_{ref} = .4$. Dilation and erosion use a 1×3 rectangular structuring element. Using the distance-based approach, they are not dual, as $\overline{\delta(\bar{u})} \neq \epsilon(u)$. This prevents the definition of self-dual morphological operators based on erosion and dilation as ones proposed by Heijmans [53]. However, as soon as v_{ref} is defined as an extreme value (black or white typically), the duality is preserved but then, the vectors are simply ordered by their norms.

u	.3	.4	.6		$\epsilon(u)$	.4	.4	.4
$\bar{u}$	.7	.6	.4		$\delta(\bar{u})$	.7	.7	.6
					$\overline{\delta(\bar{u})}$	.3	.3	.4

Figure 29: Problem of duality between erosion and dilation using the distance-based ordering. $v_{ref} = 0.4$.

DISTANCE-BASED SELF-DUAL MORPHOLOGICAL OPERATORS In [66, 48], distance-based dual erosion and dilation are defined using two reference points $C^{+\infty}$ and $C^{-\infty}$ (typically black and white) leading to two different distance functions:

$$v \leq_1 w \Leftrightarrow d(v, C^{-\infty}) \leq d(w, C^{-\infty})$$
$$v \geq_2 w \Leftrightarrow d(v, C^{+\infty}) \leq d(w, C^{+\infty})$$

Then, the infimum is defined using $\leq_1$ ([48] brings a solution to the multiple extrema pitfall and ensure the uniqueness of the minimum) while the supremum using $\geq_2$, that enables a dual definition of the erosion and the dilation. However, as noted by Goutali et al. [48], imposing a total ordering on colors is not sufficient to get self-dual morphological operators. From the color erosion and dilation operators, one can get openings and closings and then self-dual morphological operators [53]. However,

openings and closings defined this way are generally not idempotent. Indeed, we have to ensure for any set of values that the minimum extracted by the erosion is the maximum extracted by the dilation. Goutali et al. [48] proposed a solution to this problem but at the cost of the loss of the duality property. The idea of having two different ordering relations to extend the ToS to multivariate data seems difficult as it would imply to use different relation for upper and lower sets.

RANK TRANSFORMATIONS AND SPACE FILLING CURVES The distance-based ordering seen above belongs to a wider class of ordering methods based on rank transformation. More formally, a rank transformation is a function $\mathbb{R}^n \to \mathbb{N}$ that maps a vector to a rank position (so this is highly similar to the mapping h defined previously).

It has been shown that there is actually an equivalence between a total ordering on $\mathbb{R}^n$, a *bijective* mapping h, a rank transformation, and also space filling curves. In [74], the authors proposed imposing a total order using a non-linear reduction of the color space on a single dimension. Their approach based on manifold learning assumes that two close points in the original high-dimension space are projected to close locations and that local distances are preserved. They later introduced in [71] the ordering of color patches. The same idea is used but the construction now involves the spacial relationship and so the correlation between colors of two neighboring pixels is taken into account when designing the ordering.

Another example of rank transformation is the bit-mixing ordering that uses the binary representation of the vectors to deduce a single scalar value. The bits of the components of the vector are sorted, most-significant bits come first, then the second bit of the components... [34]

Rank transformation can also be interpreted in terms of space filling curves as they yield curves in the high dimensional space that attains each value exactly once. As a consequence, Z-filling curve, Peano curves... are also possible ways to define total vector ordering. In [34], theses different approaches are compared in terms of the neighboring conservation, *i.e.*, we want to avoid high distance gaps between two consecutive values in the order. This idea has been more recently retained by Chevallier and Angulo [35], where they define their objective ordering relation as an optimization problem. They aim at finding the permutation of values that minimizes the length of the curves that reach every point of the co-domain (not the full value space). However, such an optimization problem is difficult to solve and only an approximated solution can be computed.

4.3.3 *P-orderings*

Conditional and reduced orderings are the most used methods to impose a total order on a vector space. For mathematical morphology, few authors relied on P-orderings. The latter consists of clustering the vectors into groups so that they can then be ordered according to their extremeness w.r.t. the rest of the data. Velasco-Forero and Angulo [126]

rely on the random projection depth which assigns for each vector its degree of centrality in the dataset and provides a "center-outward" ordering of the data.

4.3.4 *The color "no free lunch" theorem*

We have seen in section 4.3.2, methods that try to find a space filling curve minimizing the irregularities along the path. However, the topology of a high dimensional space cannot be reproduced on a single dimension. This is highlighted in [35], with the following theorem:

Theorem 1. *For any total order* $\leq$ *on* $\mathbb{R}^n$ *(*$n > 1$*).*
$\forall R, r \in \mathbb{R}^+$, $\exists a, b, c \in \mathbb{R}^n$ *such that:*

- $a \leq b \leq c$
- $d(a, b) > R$
- $d(a, c) < r$

A direct consequence of this theorem is that for any total ordering on a vector space, there will exist two consecutive values with a large distance gap between them. This theorem encourages the use of image dependant total order. However, if we now consider the order computation as part of the morphological filter, it will lose the idempotence property, as the ordering relation used for the first filter computation may not be the same as for the second computation.

4.3.5 *Local orderings and "pseudo"-morphological filters*

Because of the consequences of theorem 1 (*i.e.*, there does not exist a-priori any relevant total ordering on a multi-dimensional value space), authors have been involved in developing image content dependant ordering relations (see. sections 4.3.2 and 4.3.3). This idea can be developed further by considering an ordering relation which is locally dependant on the data. In [72, 73], authors impose a total order for each pixel, relying only on the vector values in a spacial local window. Those values are represented as a fully connected graph where edges are valuated with a distance color. Then, they aim at extracting an Hamiltonian path (*i.e.*, a linear order) from this graph using minimum spanning trees. Yet, a drawback of a local ordering lies in that the fact that it does not provide a partial ordering of the full value set; erosion and dilation lose important algebraic properties (e.g. they do not commute with the infimum/supremum) that prevent the construction of "real" morphological filters.

4.3.6 *Component trees processing based on total (pre)orders*

Once a total order has been defined on values, computing component trees is straightforward. Any of the approaches described above can be used, even if most authors have used the most "standard" methods to obtain the ordering. In [104], the max-tree is computed on multiband images using the lexicographical order on the data where the channels have been permuted w.r.t their signal to noise ratio. In [89], authors compare lexicographical ordering and distance-based (pre)orderings with a marginal processing in the context of image filtering and document binarization. Another comparison is provided in [123], where the authors perform an evaluation of max-tree based image compression. They compare some preorders either based on luminance, chromaticity or saturation in different colorspaces.

While the construction of the tree is straightforward once the total ordering established, the restitution phase is more challenging in the case of preorders. Indeed, when the rank transformation is not injective, several vectors may map to the same value; thus a node in the tree may be associated with several vectors. However, when filtering and reconstructing with component trees (see. section 3.1.2), one has to select a single value to set for the nodes that are removed. Naegel and Passat [89] proposed several restitution strategies to solve this problem. It generally consists in choosing a representative value based on the different vectors contained in a node.

p_{mean} Each node is assigned to a representative value which is the mean vector.

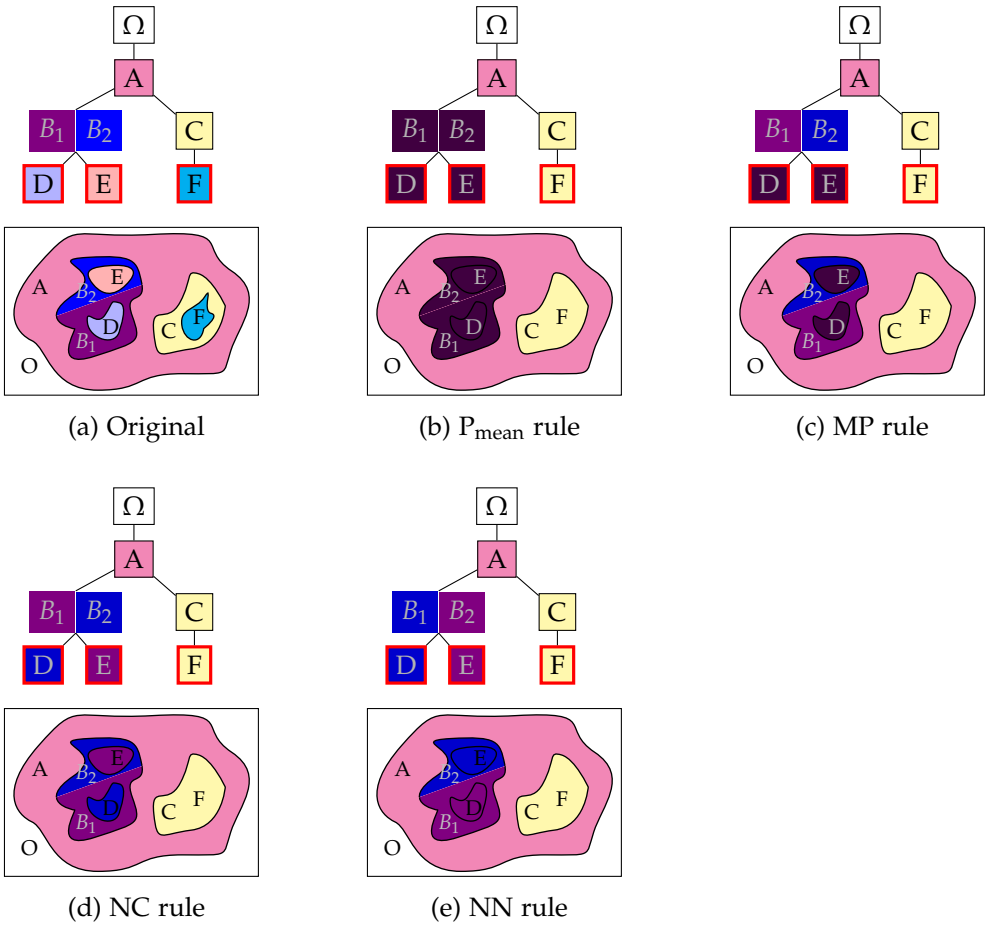
p_{median} Each node is assigned to a representative value which is the median vector (the ordering relation is completed with a lexicographical cascade to ensure antisymmetry).

As noted by [123], those strategies tend to lower the quantization of the image even if no filtering is performed. Pixels with an "equivalent" rank are altered and merge with the same "color". To overcome this issue, Tushabe and Wilkinson [123] proposed altering only pixels that belong to nodes to be removed and keeping the others to their original value. Theses are respectively referred as *Mean of Parent (MP)* and *Median of Parent*. They also proposed two other restitution strategies:

NEAREST COLOR (NC) A removed node gets assigned with the closest vector *(in the value space)* from its own mean vector in the last surviving ancestor. The pixels from non-filtered nodes keep their original value.

NEAREST NEIGHBOR (NN) A pixel in a removed node gets assigned with the value of the nearest pixel *(in the domain space)* of the last surviving ancestor. The pixels from non-filtered nodes keep their original value.

Note that the P_{mean} and *Nearest Color* strategies are subject to the "false color" problem since the average vector may not exist in the original image spectrum. On the other hand, the *Nearest Neighbor* strategy yields a "pseudo"-connected filtering as a removed node



(a) Original (b) P_{mean} rule (c) MP rule

(d) NC rule (e) NN rule

Figure 30: The different restitution strategies on the ToS when using a preorder on values. The nodes with red borders are those to filter.

may be split and reconstructed with regions of different values. The later was motivated by the authors who wanted to reach a filtering favoring a better image regularity with smooth edges along the removed components. The different strategies are illustrated on fig. 30 where data are ordered by luminance (so a preorder). The regions B_1 and B_2 have the same rank so they belong to the same node in the ToS. With the P_{mean} strategy B_1B_2, D and E are filtered with the same value (the average color of B_1 and B_2) while with the MP rule, B_1 and B_2 remain unchanged. In both cases, a new color is introduced. This is not the case with the NC and NN rules where B and E are filtered with the value b_1 and b_2. We can also notice that on this scheme, the objective of the NN rule is reached, the regions to remove have weak edges along their boundaries.

4.4 CONCLUSION

In this chapter, we have exposed the most well-known approaches to process multivariate images with the ToS. We have seen that marginal processing is unsatisfactory as it yields the management of multiple trees at the same time, and we do not have finally a single structure representing the image. As a consequence, we have considered the vectorial approach to extend the ToS through the imposition of a total order. We have shown many strategies to impose a total order. Some of them try to impose an ordering on the full value space, but we highlighted the fact that any ordering of this type will have strong discontinuities. As a consequence, we have reviewed other types of ordering that are image-dependant or locally-dependant but we also highlighted that if the ordering computation is considered as a part of the process, usual self-dual morphological filters based on them may lose important properties such as the idempotence, or the self-duality. This motivates the claim that we would be better to find a way to extend the ToS to multivariate images without trying to impose a total order on values.

5 COMPONENT GRAPHS

In chapter 4, we have seen a classification of the methods imposing an ordering relation on a vector space. Since a natural total order is not obvious, Passat and Naegel [102] were the first to propose an extension of max-trees employing the M-ordering approach, that is an ordering comparing the components channel-wise and yielding a *partial* order. As a consequence, two values may not be comparable, components may overlap without being nested. The underlying structure to process is no more a tree but a graph. In the section 5.1 of this chapter, we recall the method extending max-/min- tree to partial order proposed by Passat and Naegel [102] and show that it can be easily adapted for the ToS. In section 5.2, we will also expose the way of processing these graphs and have a word about the algorithmic complexity.

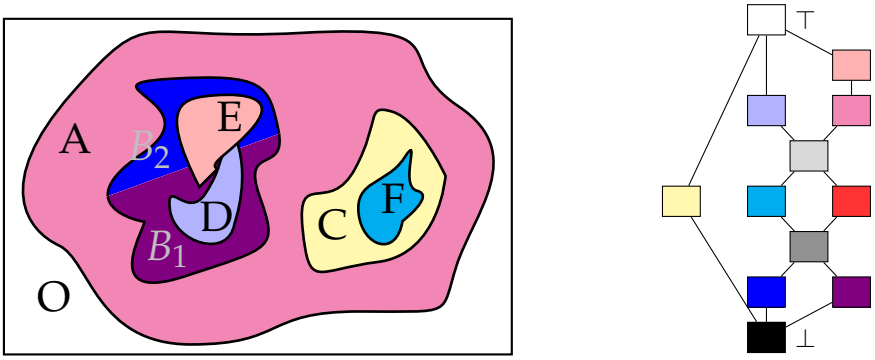
5.1 COMPONENT TREES EXTENDED TO PARTIAL ORDER

Let u an image from $\Omega \to F$ where F is endowed with an ordering relation $\leq$. We have seen in section 3.1 that if $\leq$ is total, the sets of lower connected components Γ^- or upper connected components Γ^+ form a tree. Now that $\leq$ is partial (we note it $\preceq$ now), two components may overlap without being nested. As a direct extension of the definition of the max-tree (resp. min-tree), [102] proposed to consider the Hasse diagram $\dot{\mathfrak{G}}$ of the cover of $(\Gamma^+, \subseteq)$ (resp. $(\Gamma^-, \subseteq)$) which now forms a graph. If $\preceq$ is total, the component graph is obviously equivalent to its component tree. The same idea can be applied to extend the ToS as one just has to consider the cover of $(\mathcal{S}, \subseteq)$ which also forms a graph.

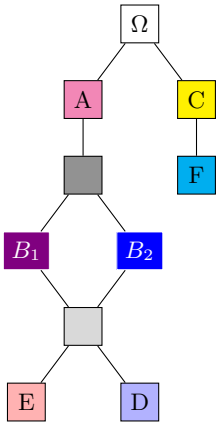
[102] also proposed an alternative definition for the component graph $\ddot{\mathfrak{G}}$ which only considers the connected components that are sup/inf-generators of u. More formally:

$$\ddot{\Gamma}^+ = \bigcup_{\lambda \in F} \{\, X \in \mathcal{CC}([u \succeq \lambda]) \mid \bigwedge u(X) = \lambda \,\}$$
$$\ddot{\Gamma}^- = \bigcup_{\lambda \in F} \{\, X \in \mathcal{CC}([u \preceq \lambda]) \mid \bigvee u(X) = \lambda \,\}$$

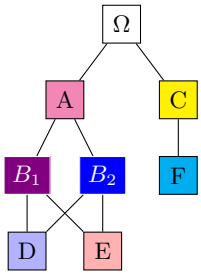
From an algorithmic point of view, it simply consists in removing the "empty" nodes from the graph $\dot{\mathfrak{G}}$. Note that in [102], the authors propose also a third graph where two same connected components issued from two different threshold sets are represented by two disjoint nodes in the graph. While this is useful as a matter of clarity and theory, we do not consider this representation as it implies many redundancies and leads to a representation which is not equivalent to component trees in grayscale.



(a) Original image and the Hasse diagram of $(F, \preceq)$



(b) Graph $\dot{\mathfrak{G}}$



(c) Graph $\ddot{\mathfrak{G}}$

Figure 31: Example of ToS extensions using the approach in [102]. (b) is the first graph $\dot{\mathfrak{G}}$ using every possible cuts. (c) is the alternative graph $\ddot{\mathfrak{G}}$ using only sup/inf-generating connected components.

In fig. 31, we show how this approach can be used to extend the ToS on multivariate images. We use the same input image as in fig. 30 but now B_1,B_2,D and E all connect to each other for the sake of the illustration. In the first definition of the extension, nodes with "false" colors appear in the graph $\mathfrak{G}$ (those in grays, see. fig. 31b). They correspond to the supremum/infimum of the colors of connected components that are not comparable. Those nodes are actually "empty" has they do not contain any proper pixel that is not in one of their children. On the other hand, the second formulation prevents the apparition of "false" colors in the graph $\ddot{\mathfrak{G}}$ as only cuts from value existing in the original image are considered.

ALGORITHMIC AND COMPLEXITY CONSIDERATIONS An early algorithm to compute the component graph (in the context of extending min and max-trees) was proposed in [90] and then corrected in [88]. As noted by the authors, algorithms used to compute the

component trees on grayscale cannot be transposed to partial orders. An exception is if the order has a particular lattice structure (namely a lower/upper piecewise total order), where the component-graph is actually a hierarchy [103, 63]. The algorithm proposed by Naegel and Passat [88] to compute $\ddot{\mathfrak{G}}$ is kind of "brute-force":

1. Compute every connected component of every threshold set $[u \preceq \lambda]$ for all $\lambda \in F$,
2. Compute the graph of inclusion of the components,
3. Compute the transitive reduction of the graph.

If we consider the standard RGB space encoded on 24 bits, this implies testing every 2^{24} cuts, which is computationally untracktable. In the special case of $\ddot{\mathfrak{G}}$, the authors propose a dedicated algorithm that does not depend on the size of the value space [88]. Yet, its complexity is still high ($O(|\Omega|^2)$), and the authors tackle the problem by processing locally small patches and merging them back. While this is sensible for filtering, it is not adapted for applications that need a representation of the full image (such as object detection). In addition, this approach is not easily extendable to the ToS since connected components need to be hole-filled first. A straightforward adaptation of the algorithm would lead to a complexity $O(|\Omega|^3)$.

5.2 PROCESSING THE COMPONENT GRAPHS

As for grayscale component trees, the image can be reconstructed using the same formula eq. (3.1) for min- and max- component graphs and eq. (3.3) for the graph extension of the ToS. Equivalently, the same filtering strategies exposed in section 3.2.3 are available for the component graph. However, as a node may now have several parents, the min strategy is ambiguous. In [102], it is declined in two strategies:

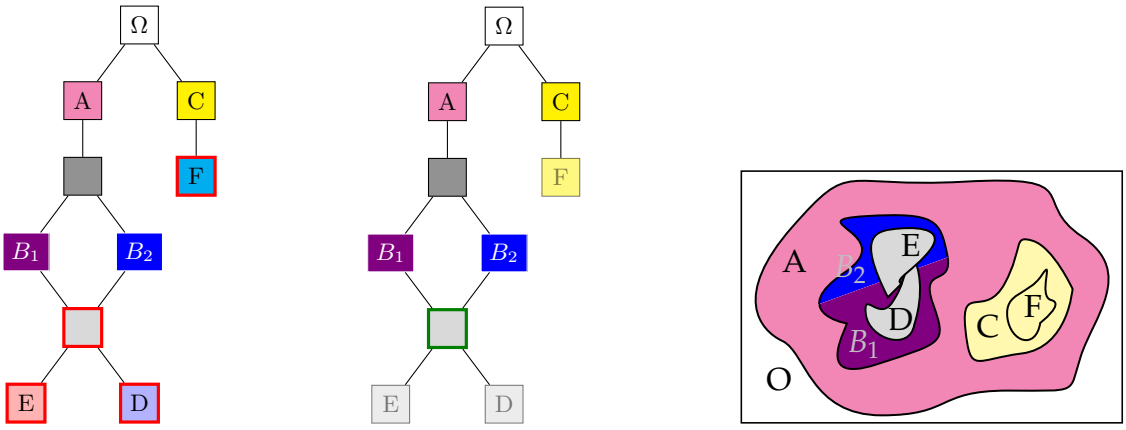
MIN$_1$ A node is kept if it passes the criterion and *all* the path from this node to the root are "alive".

MIN$_2$ A node is kept if it passes the criterion and *there exists* an "alive" path from this node to the root.

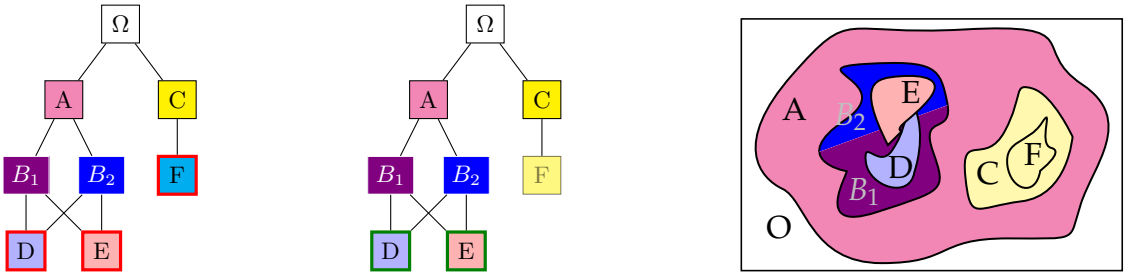
The fact that a node may have several parents yields another problem with the reconstruction from a filtered tree. Consider Γ the initial set of components/shapes, $\Gamma' \subset \Gamma$ the set of components/shapes in the filtered graph, and the operator $\cap_{\mathcal{CC}}(A, B) = \mathcal{CC}(A \cap B)$ which returns the connected components of the intersection of two components. Then, $\Gamma \cup \{\emptyset\}$ is stable w.r.t $\cap_{\mathcal{CC}}$ (*i.e.*, $\forall A, B \in \Gamma$, $\cap_{\mathcal{CC}}(A, B) \subset \Gamma$), while $\Gamma' \cup \{\emptyset\}$ is not. In other words, a point may belong to several nodes. Naegel and Passat [90] tackle the problem with a coherence recovery step which consists of adding in Γ' some nodes of Γ until reaching the stability in Γ'. This is illustrated in fig. 32. The node F has a single "alive" parent, so the restoration process is the same as for the component trees. It takes the value of its parent. On the other hand, the nodes D, E and DE have two "alive" parents,

we have a conflict with the value to propagate. The coherence recovery steps aim at recovering the highest nodes such that the reconstruction could be performed.

Even if the filtering with component graphs is $O(N)$ where N is the number of nodes, the coherence recovery process adds an extra-complexity compared to trees. In addition, the computation of attributes is not as trivial. Attribute computation is a bottom-up incremental process and one has to ensure that a node will not be counted twice when it has several paths leading to the root.



(a) Filtering and coherence recovery on $\dot{\mathfrak{G}}$



(b) Filtering and coherence recovery on $\ddot{\mathfrak{G}}$

Figure 32: Coherence recovery with component graphs. Left: original component graph, red nodes are the ones to remove. Middle: graph after the coherence recovery step, green nodes are the recovered ones. Right: image reconstruction.

5.3 CONCLUSION

In this chapter, we have exposed the innovative approach proposed by Passat and Naegel [102] to extend the min- and max-trees to partial order so that we do not have to choose a particular total order. We have shown that the same principle could be apply to extend the ToS. However, because the ToS is based on hole-filled components, it requires extrawork and the algorithms they propose cannot be adapted easily. In addition, those algorithms

are so computationally expensive that it compels the authors to process the image by patches. The latter has an important disadvantage for applications that require having a representation of the full image. We also highlighted a second disadvantage of the method in that it produces a graph and no longer a tree. Some basic processing that is simple with trees becomes more tricky with graphs, such as the incremental computation of attributes or the filtering that requires an extra coherence recovery step. Moreover, the existing morphological state-of-the-art methods about object detection, simplification, etc., are dealing with trees. This point justifies the claim that we would better have a tree to process than a graph.

Part II

A NEW APPROACH FOR A TREE OF SHAPES ON MULTIVARIATE IMAGES

6

THE MULTIVARIATE TREE OF SHAPES (MTOS)

As seen in section 3.2, the ToS [32, 28] is a hierarchical representation of the image in terms of the inclusion of its level lines. Its power lies in the number of properties held by this structure that one would need for efficient image processing and computer vision as discussed in [21]. Indeed, some authors have already used it successfully for image processing and computer vision applications. In [43, 13, 139, 142], authors use an energy optimization approach on the ToS hierarchy for image simplification and image segmentation while in [21] relies on an a-contrario approach to select meaningful level-lines. Other applications include blood vessel segmentation [144], scene matching extending the MSER through the Maximally Stable Shapes [28] and the Tree-Based Morse Regions [143], image registration [28], multispectral image classification [33]. . . for which it competes with the state-of-art methods in their respective field. Apart from being effective, the most remarkable things about the ToS are its versatility and its simplicity. We have cited some examples of applications involing the ToS which spread over the whole domain of the computer vision, but we did not yet say a word about the way of processing this tree. The abstraction offered by the ToS enables us to perform advanced image processing tasks in a simple way [26]. For example, an image simplification can be performed by selecting or removing some nodes and a denoising by pruning some branches of the tree as seen in section 3.2.3.

It is not so much a surprise that the ToS achieves such good results, but it is rather due to the mathematical properties held by the representation [20]. The ToS is the support for self-dual, contrast invariant and morphological connected operators.

- First, it is a morphological representation based on the inclusion of the connected components of the image at different level of thresholding. As such, a basic filtering of this tree is a connected filter that is an operator that does not move the contours of the objects but only keep or remove some of them [112].

- Second, it is invariant by any contrast change. Yet it is not invariant to illumination change but robust to *local* change of contrast as we expect the level lines to remain globally the same. This property is very desirable in many computer vision applications where we face the change of illuminantion problem e.g. for scene matching, object recognition. . . In fig. 33b, we show this invariance by simulating a change of illuminantion directly in the ToS; so we have the exact same tree representation as for the original image in fig. 33a.

- Third, besides being contrast change invariant, the ToS is also a self-dual representation of the image. This feature is fundamental in a context where structures

may appear both on a brighter background or on a lighter one. Therefore, auto-dual operators are particularly well-adapted to process images without apriori knowledge on the content disposition. While many morphological operators try to be self-dual (e.g. Alternating Sequential Filters) by combining extensive and anti-extensive operators, many of them actually depend on the processing order (*i.e.*, on which filter comes first). Self-dual operators have the ability to deal with both dark and light objects in a *symmetric* way [53, 117] (see fig. 33b).

- Fourth, it allows a multiscale analysis of the same. Contrary to many key point detectors (e.g. Harris corners detectors...) which are highly local (thus the need for a scale space in the SIFT method), the level lines (thus the shapes) are more global as they are closed curves that can spread on the whole image. Multiscale analysis is also very simple and efficient since it is encoded through the inclusion of the level lines embedded in a tree structure which is both fast to compute and easy-to-process [46, 140]. In fig. 33c we show that the level lines of the same object taken from different views (different scale and different illumination) correspond.

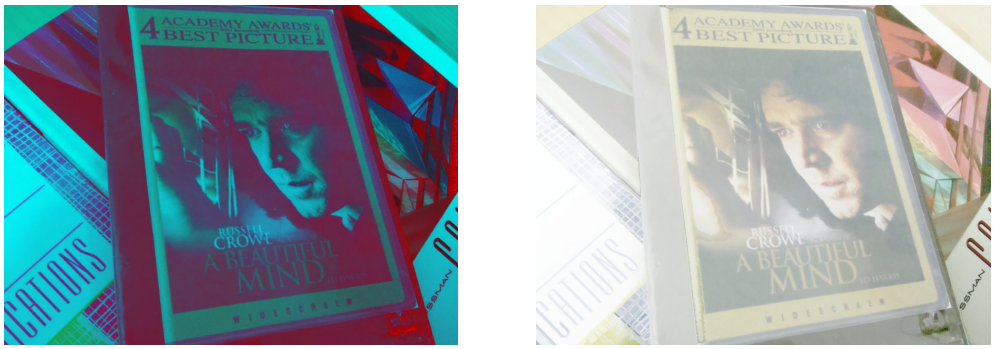
- Last but not least, the level lines fit the object boundaries as they are located everywhere in the image, tangent to the gradient. Contrary to many key-point detectors which rely on local information, level lines may be large Jordan closed curves that spread over the entire image. It is thus well-adapted for scene analysis and content extraction.

While the ToS is well-defined on grayscale images, it is getting more complicated with multivariate data. Indeed, like most morphological trees (e.g. min and max-trees), the ToS relies on an ordering relation on values which has to be total. If it is not, the definition based on lower and upper cuts yields components that overlap and the inclusion tree is ill-formed.

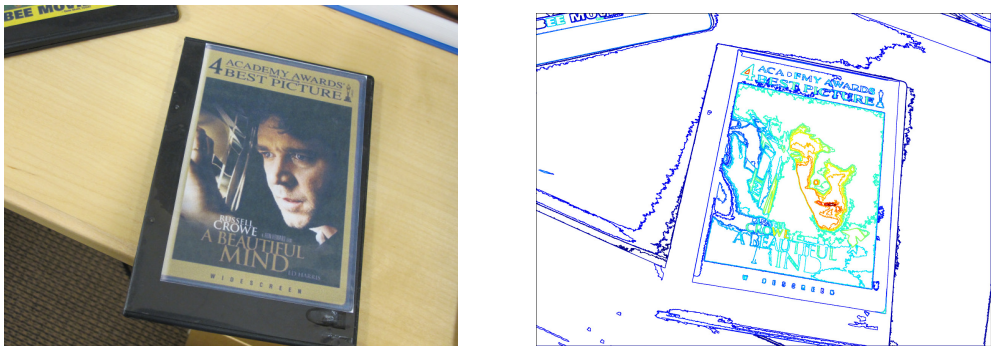
However, the problem of correctly handling multivariate data is of main interest, since they arise in many image processing fields. The most well-known example is the sensitive color processing of natural images, but many other types of images are multivariate: satellites provide multispectral or hyperspectral images with hundreds of bands, medical processing provide multimodal images acquired by several devices... Therefore, to overcome this problem, most authors have been focusing on defining a total order on multivariate data as exposed in chapter 4. However, from our point of view, the most important concept in morphological trees lies the inclusion of shapes. As a consequence, we introduce a novel approach which does not intend to build a total order, but tries to build up a set of non-overlapping shapes from an arbitrary set of shapes using the inclusion relation only.

This chapter is organized as follows. In the section 6.1, we explain the main lines of the method extending the ToS to handle multivariate data. In section 6.2, we expose the Graph of Shapes (GoS) as a structure merging several ToS's, and in section 6.3, we explain how we compute a tree from it.

(a) A film cover and its meaningful level lines extracted with the MToS. Level lines are selected using the Maximum Stability criterion (as for the MSER) and colorized w.r.t their level of inclusion.



(b) On the need for contrast change/inversion invariance. The original image has been subjected to a global marginal inversion and/or change of contrast (left) and to a local change of contrast (right). They give the same MToS, so the same level lines depicted in (a).



(c) A different view of the image (a). The level lines are selected with same method and globally match the ones of (a).

Figure 33: Robustness of the level lines w.r.t tranformations in the value space (b) and the domain space (c).

6.1 METHOD OVERVIEW

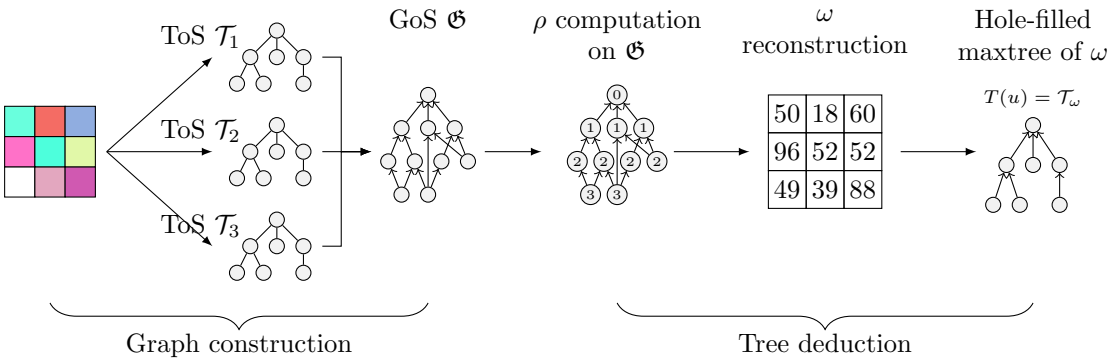


Figure 34: The 5-step process of the proposed method. (1) The input image u is decomposed into individual channels $u_1, u_2, \dots u_n$, for which the ToS's are computed, (2) the ToS's are merged into the GoS $\mathfrak{G}$ (2), an algebraic attribute is computed on $\mathfrak{G}$ and, (4) yields a scalar attribute map ω, (5) a final tree is built upon ω.

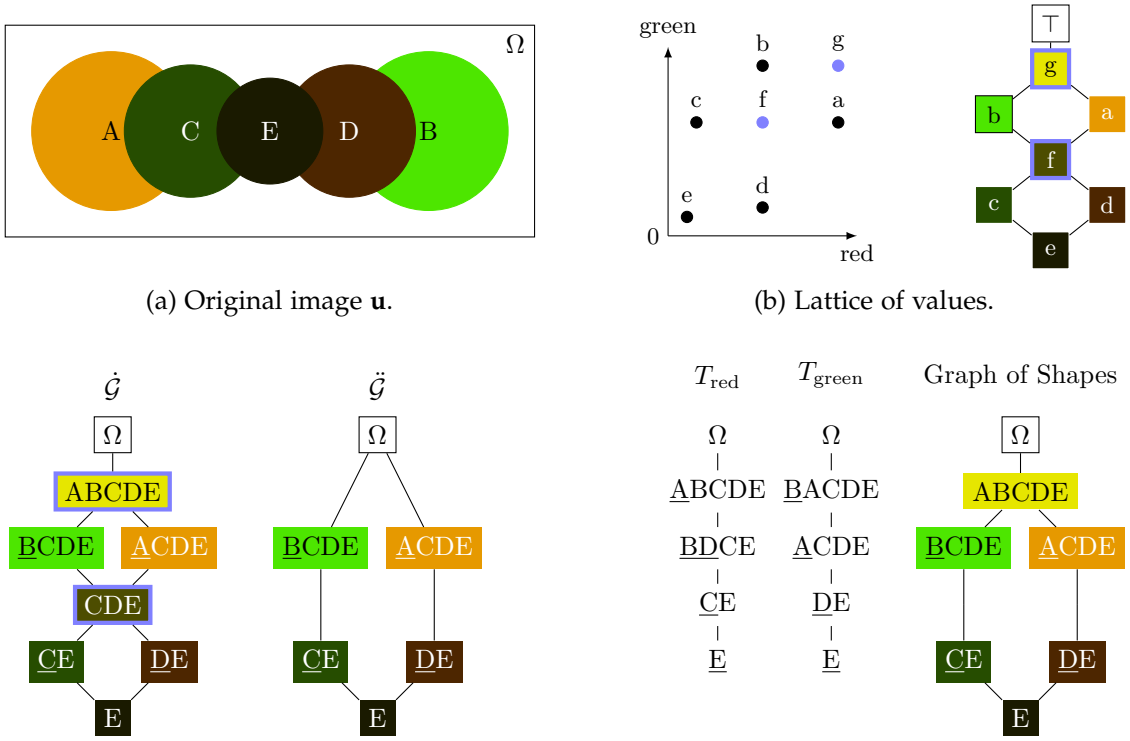
In section 3.2, we gave the definition of *shapes* for gray-level images as hole-filled connected components from lower and upper threshold sets of u. Let us first relax this definition. A *shape* X is a connected component of Ω without holes (*i.e.*, such that $\mathcal{H}(X) = X$).

Given a family of shape sets, namely $\mathcal{M} = \{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n\}$, where each element $(\mathcal{S}_i, \subseteq)$ forms a tree, we note $\mathcal{S} = \bigcup \mathcal{S}_i$ the initial shape set. Note that $(\mathcal{S}, \subseteq)$ generally does not form a tree but a graph since shapes may overlap. We aim at defining a new set of shapes $\boldsymbol{S}$ such that any two shapes are either nested or disjoint. We do not constrain $\boldsymbol{S} \subseteq \mathcal{S}$, *i.e.*, we allow the method to build new shapes that were not in the original shape set. We note $T : \mathbb{R}^{n\Omega} \to (\mathcal{P}(\mathcal{P}(\Omega)), \subseteq)$ the process that builds a tree of shapes $(\boldsymbol{S}(\mathbf{u}), \subseteq)$ from an image $\mathbf{u} \in \Omega^{\mathbb{R}^n}$.

The method we propose is a simple 5-step process which consists basically in two parts depicted in fig. 34. The first part is the construction of a GoS from ToS's computed marginally on each component (steps 1–2). The second part aims at deducing the tree from the GoS and consists in computing a tree over an image reconstructed from an attribute valuated on the GoS (steps 3–5).

6.2 THE GRAPH OF SHAPES (GOS)

First $\mathbf{u}$ is decomposed into its individual channels $u_1, u_2, \dots, u_n$ for which the ToS's $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$ are associated with the shape sets $\mathcal{S}_1$, $\mathcal{S}_2$, …, $\mathcal{S}_n$. Let $\mathcal{S} = \bigcup \mathcal{S}_i$, we call the GoS $\mathfrak{G}$ the cover of $(\mathcal{S}, \subseteq)$, *i.e.*, it is the inclusion graph of all the shapes computed marginally.



(a) Original image **u**.

(b) Lattice of values.

(c) "Shape" component-graphs $\dot{\mathfrak{G}}$ and $\ddot{\mathfrak{G}}$.

(d) Marginal ToS's and the GoS $\mathfrak{G}$.

Figure 35: Comparison of the GoS $\mathfrak{G}$ with component graphs $\dot{\mathfrak{G}}$ and $\ddot{\mathfrak{G}}$.

By starting with computing the marginal ToS's of u, we have an initial shape set. The trees provide a representation of the original image and **u** can be reconstructed marginally from them. However, handling multiple trees is not straightforward. The trees themselves lack some important information: how the shapes of one tree are related (w.r.t. the inclusion) to the shapes of the other trees. The graph $\mathfrak{G}$ is nothing more that these trees merged in a unique structure that adds the inclusion relation that was missing previously. As consequence, $\mathfrak{G}$ is "richer" than $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, and because the transformation from $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ to $\mathfrak{G}$ is reversible, $\mathfrak{G}$ is a complete representation of **u**; *i.e.*, **u** can be reconstructed from $\mathfrak{G}$. Indeed, this is a component-wise restitution, *i.e.*, $\tilde{\mathbf{u}}(x) = (\tilde{u}_1(x), \ldots, \tilde{u}_n(x))$ where $\tilde{u}_i(x)$ is given by eq. (3.3) based on the shape set $\mathcal{S}_i$. In addition, $\mathfrak{G}$ is also a self-dual and a contrast invariant representation of **u** because $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ are.

6.2.1 *Differences with component graphs*

In chapter 5, we have introduced an extension of the ToS to partial order in the same way that Passat and Naegel [102] did with component-graphs for min- and max-trees. The GoS $\mathfrak{G}$ is different from them as illustrated in fig. 35. $\dot{\mathfrak{G}}$ is actually finer than $\mathfrak{G}$

Figure 36: A GoS that is an invalid morphological tree. Left: two marginal shapes from red and green channels are overlapping. Right: The corresponding Graph of Shapes (GoS) $\mathfrak{G}$. The points of $A \cap B$ belong to both nodes A and B, so the tree $\mathfrak{G}$ is not valid.

as any lower marginal cut can be obtained from $[u \preceq (\top, \ldots, \top, \lambda_i, \top, \ldots, \top)]$ (and upper marginal cuts as well). The shapes of $\mathfrak{G}$ is thus a subset of the shapes of $\dot{\mathfrak{G}}$. On the other hand, $\mathfrak{G}$ seem unrelated to $\ddot{\mathfrak{G}}$ which is only composed of the shapes that are inf/sup-generators of $\mathbf{u}$. $\mathfrak{G}$ has a strong interest from a computational point of view because, contrary to $\dot{\mathfrak{G}}$ and $\ddot{\mathfrak{G}}$, $\mathfrak{G}$ can be computed efficiently by merging the marginal ToS's built on individual channels.

Note also that with "standard" morphological hierachies (min-/max- trees) and their extension (the component-graph [102, 103]), for any point x, there exists a *single* smallest component that contains x. As a consequence, a point belongs to a *single* node in the structure. In the GoS, a point may belong to several nodes. For example, in fig. 36, the points in $(A \cap B)$ belong to both nodes A and B, but $(A \cap B)$ does not exit as a node in any marginal tree. This leads to a weird paradox, even if the GoS is actually a tree, it is not a valid morphological tree. Thus, we cannot just extract a tree (e.g. with the minimum spanning tree) from the GoS as it would not be valid.

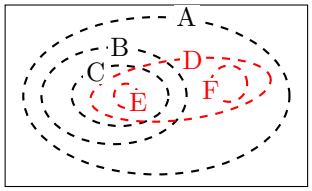
6.3 COMPUTING A TREE FROM THE GRAPH

The second part of the method tries to extract a tree from $\mathfrak{G}$. Let $\rho : \mathcal{P}(\Omega) \to \mathbb{N}$ be an algebraic decreasing shape attribute, *i.e.*,
$\forall A, B \in \mathcal{S}, A \subset B \Rightarrow \rho(A) > \rho(B)$

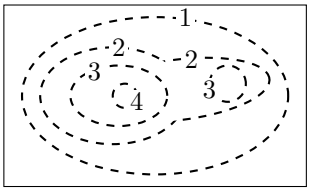
We will discuss in the next section how we choose ρ and why we consider the *depth* attribute. The *depth* of a shape in $\mathfrak{G}$ is the length of the longest path of a shape A from the root. Let $\omega : \Omega \to \mathbb{R}$ defined as:

$$\omega(x) = \max_{X \in \mathcal{S}, x \in X} \rho(X) \tag{6.1}$$

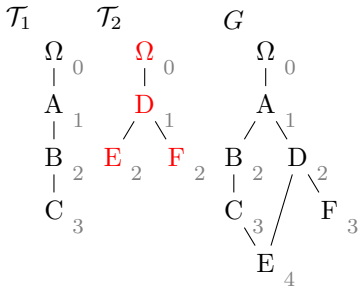
The map ω associates each point x with the depth of the deepest shape containing x . Let $\mathbb{C} = \bigcup_{h \in \mathbb{R}} \mathcal{CC}([\omega \geq h])$. $(\mathbb{C}, \subseteq)$ is actually the max-tree of ω. Finally, we consider $\mathcal{S} = \mathcal{H}(\mathbb{C})$ and $(\mathcal{S}, \subseteq)$ as the final MToS $\mathcal{T}_\omega$. The method is illustrated in fig. 37 where a two-channel image has overlapping shapes from red and green components. The ToS's $\mathcal{T}_1$ and $\mathcal{T}_2$ are computed marginally and merged into the GoS $\mathfrak{G}$ for which we compute the depth of each node (fig. 37b). Those values are reported in the image space, pixel-wise (fig. 37c). This is this step at which we decide which shapes are going to merge; here B



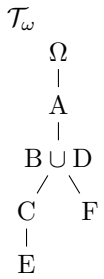
(a) Original image u (2-channels) and its shapes.



(c) ω image built from $\mathfrak{G}$



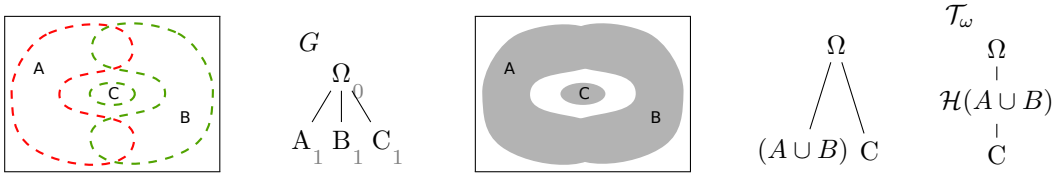
(b) The marginal ToS's $\mathcal{T}_1$, $\mathcal{T}_2$ and the GoS $\mathfrak{G}$. The depth appears in gray near the nodes.



(d) The max-tree $\mathcal{T}_f$ of ω

Figure 37: The method illustrated on an example.

and D are set to the same value. This choice is based on the level of inclusion and no longer on the original pixel values. Eventually, the max-tree of the depth map is built and yields the final MToS. In this example, the maxtree of ω gives valid shapes. However, because $\mathbb{C}$ may form components with holes, the hole-filling ensures those components are valid as shown in fig. 38.



(a) (b) (c) (d) (e)

Figure 38: On the need for hole-filling. (a) Original 2-channel image. (b) The GoS $\mathfrak{G}$ valuated with depth. (c) Depth map ω reconstructed from $\mathfrak{G}$. (d) Maxtree of ω without cavity filling. (e) Maxtree of ω with cavity filling. In (d), the max-tree gives the shape $A \cup B$ that has a hole, so we need the hole-filling step to ensure valid shapes (e).

Let us now explain the rationale of this part. The main objective was to get a new set of shapes from $\mathfrak{G}$ that do not overlap. The first observation is that for any decreasing attribute ρ, then $(\mathcal{S}, \subset)$ is isomorphic to $(S, \mathcal{R})$ where $A \mathcal{R} B \Leftrightarrow \rho(A) > \rho(B)$ and $A \cap B \neq \emptyset$. This just means that the inclusion relationship between shapes that we want

to preserve can be expressed in terms of a simple ordering relation on $\mathbb{R}$ with the values of a decreasing attribute. Suppose now that $(\mathcal{S}, \subset)$ is a tree and consider the image $\omega(x) = \max_{x \in X, X \in \mathcal{S}} \rho(x)$, we thus have $\mathbb{C} = \{\mathcal{CC}([\omega \geq h]), h \in \mathbb{R}\} = \mathcal{S}$. In other words, the max-tree of w reconstructed from ρ valuated on a tree $\mathcal{T}$ yields the same tree. More generally, if a shape A does not overlap any other shape, it belongs to $\mathcal{CC}([\omega \geq h])$.

6.3.1 *Image reconstruction from the MToS*

Whereas $\mathfrak{G}$ is a complete representation of $\mathbf{u}$, the ToS is not, so $\mathbf{u}$ cannot be reconstructed from it. Indeed, the tree construction process merges some marginal shapes (in the most sensible way as possible). Consequently a node of the final tree gets associated with multiple values of $\mathbf{u}$. Actually this problem is not new; we already faced it in section 4.3.6 when extending the min- and max- trees with a total preorder, where the loss of the anti-symmetry implies that some "equivalent" values belong to the same node. As a reminder, we have seen that Naegel and Passat [89], and later Tushabe and Wilkinson [123], have introduced some reconstruction strategies to solve the assignment problem. The main idea is to associate to a node a single value computed from the set of values it contains. For example, in [89] the authors proposed to assign the average vector or the median vector to the node. In [123], the same principle is applied but only to the pixels that belong to the nodes that are filtered out; the values of the other pixels remained unchanged. The authors also proposed two others strategies, assigning the closest pixel's value from the last surviving parent, where "closest" can be interpreted in the value space (1$^{\text{st}}$ strategy) or in the domain space (2$^{\text{nd}}$ strategy). The interested reader is referred to those papers and to section 4.3.6 for more details. Unless otherwise specified, in part iii, to reconstruct an image from a MToS, we use the strategy that assigns to each node the average vector value from the original image.

6.4 CHOOSING A SENSIBLE ρ FUNCTION

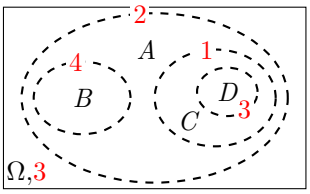
The 3$^{\text{rd}}$ step of the method involves choosing an attribute to be computed over the GoS $\mathfrak{G}$. This is a critical step since it decides afterward which shapes are going to be merged or removed.

6.4.1 *Level-Lines as a Distance Problem*

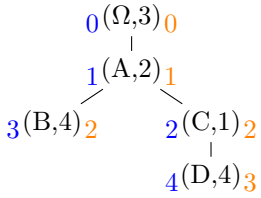
Consider the distance between two points (p, p') in Ω:

$$d_{\text{TV}}(p, p') = \min_{C_{(p,p')}} \int_0^1 |\nabla u(C(t)).\dot{C}(t)|\, dt, \tag{6.2}$$

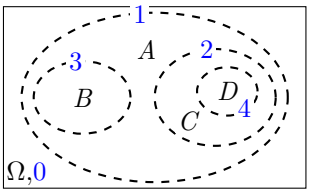
where $C(t)$ is a path in Ω from p to p'. Equation (6.2) is actually the minimum total variation (TV) along a path between p and p'. This measure has been used by Dubrovina



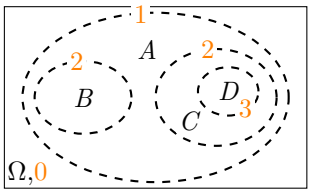
(a) Original image u, its shapes and level lines.



(b) The ToS of u and the valuation of ρ_{TV} (blue) and ρ_{CV} (orange).



(c) The level lines of ω_{TV}.



(d) The level lines of ω_{CV}.

Figure 39: Equivalence between the level lines of a gray-level image u and the level lines of the distance maps ω_{TV} and ω_{CV}.

et al. [44] for segmenting where the ToS is used as a front-end to compute efficiently the level set distance. Let $\omega_{\text{TV}}(x) = d_{\text{TV}}(\partial\Omega, x)$ be the Total Variation distance map from the border. This distance map can be computed using a simple decreasing attribute on the ToS by summing the variations from the root to the nodes. Then, instead of considering the tree $\mathcal{T}$ of u level lines, one can consider the max-tree $\mathcal{T}_\omega$ of equidistant lines. Both are equivalent in gray-level (by prop. 5).

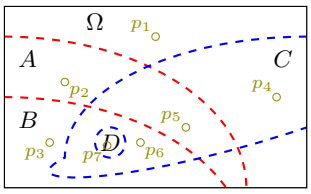
The problem with the Total Variation metric lies in that it depends on u, *i.e.*, ω_{TV} is not contrast invariant. A contrast invariant counterpart would be to only count the number of variations, *i.e.*, the minimum number of level lines to traverse to get p:

$$d_{\text{CV}}(p, p') = \min_{C_{(p,p')}} \int_0^1 \mathbb{1}\{\nabla u(C(t)).\dot{C}(t)\}\, dt. \tag{6.3}$$

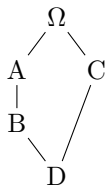
Algorithmically speaking, building ω_{CV} consists of computing the depth attribute $\rho_{\text{CV}}(A) = |A\uparrow|$ and reconstructing $\omega_{\text{CV}}(x) = \max_{X \in \mathcal{S}, x \in X} \rho_{\text{CV}}(X)$. This process is shown in fig. 39.

6.4.2 *Distance Map with Multivariate Images*

Based on the equivalence between level lines and equidistant lines in gray levels, one can produce a distance map for multivariate images. As with eq. (6.3) the idea is to count the number of marginal level lines to traverse. Depending on the way we count the level lines, the distance map may have several semantics:

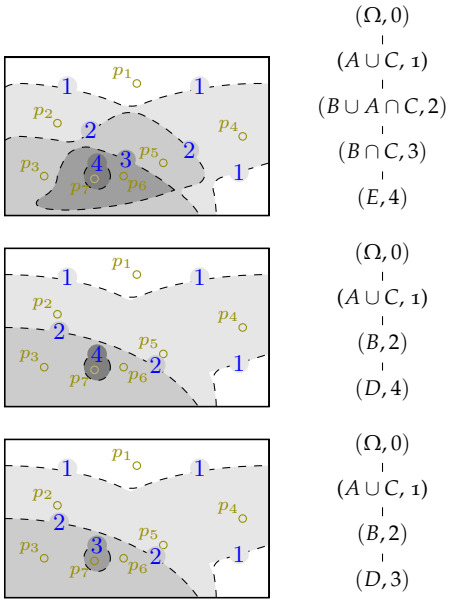


(a) Original image u, its shapes and level lines.



	D_1	D_2	D_3
p_1	0	0	0
p_2	1	1	1
p_3	2	2	2
p_4	1	1	1
p_5	2	1	1
p_6	3	2	2
p_7	4	4	3

(b) Graph of shapes $\mathfrak{G}$ and measures D_1 D_2 and D_3 valuated on markers.



(c) The distance maps ω computed with D_1, D_2 and D_3 (from top to bottom) and their corresponding max-trees on the right.

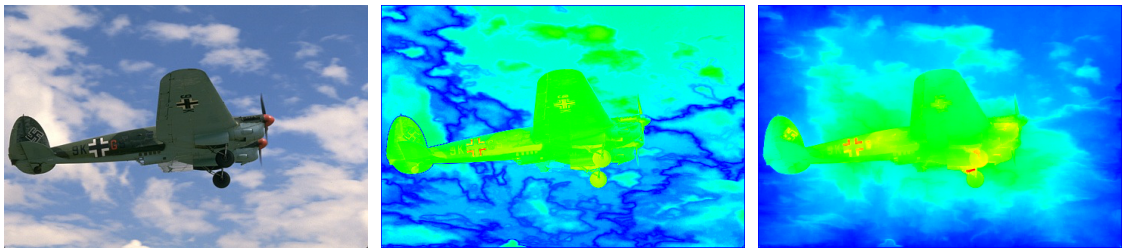
Figure 40: Differences between D_1, D_2, D_3 for the distance map computation.

d_1: $\omega_{\mathrm{CV}}(x) = |X \in \mathcal{S}, x \in X| - 1$
Count the minimal number of marginal level lines we need to traverse to get x from the border.

d_2: $\rho(A) = |A\uparrow|$ and $\omega_{\mathrm{CV}}(x) = \max_{X \in \mathcal{S},\, x \in X} \rho(X)$
Count the number of marginal level lines to traverse to get the deepest shape that contains x.

d_3: $\rho(A) = \max_{\phi \in [\Omega \rightsquigarrow A)} |\phi|$ and $\omega_{\mathrm{CV}}(x) = \max_{X \in \mathcal{S},\, x \in X} \rho(X)$
where $[\Omega \rightsquigarrow A)$ stands for the set of paths from the root to A in $\mathfrak{G}$. It counts the number of marginal level lines *that are nested* to traverse to get the deepest shape that contains x.

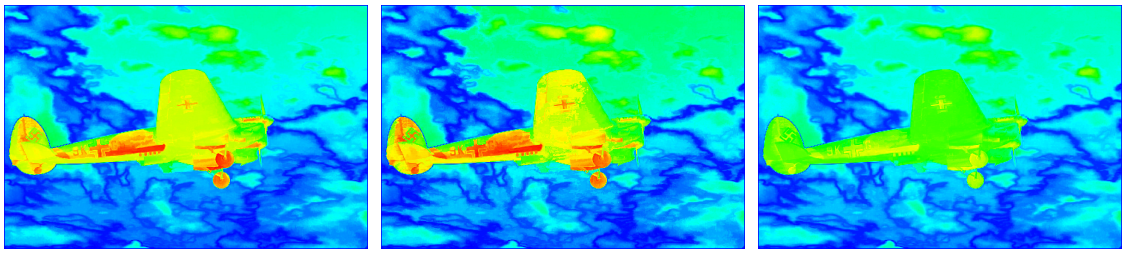
These measures can be computed efficiently from $\mathfrak{G}$ using basic graph algorithms (*e.g.*, shortest path algorithm for D_3 and ancestors counting algorithms for D_1 and D_2). The differences between them are shown in fig. 40. While, the D_2 and D_3 distances yield a union of shapes only, the D_1 distances enables getting both union and intersection of shapes. However, in practice, they define similar shape sets that differ essentially for very small components (*i.e.*, at the noise level) as shown in fig. 41. Indeed, figs. 41d to 41f show that the level lines of the distance maps computed with the three measures are quite similar and they agree with the level lines of the gray level version of the original image



(a) Original image

(b) Level lines of the TV distance computed on a gray level version of (a)

(c) Distance map computed with a Dijkstra-like algorithm.



(d) ω map computed with D_1

(e) ω map computed with D_2

(f) ω map computed with D_3

Figure 41: Distance maps computed with D_1, D_2, D_3 on a practical example. Distances are shown with the heat LUT for a better understanding. While (d), (e), and (f) give a set of level lines that agree with (b), (c) does not.

(fig. 41b). As a consequence, for the illustrations in part iii, we consider the distance D_3 as it is the fastest to compute. Note that it might be tempting to compute the distance map ω using a more conventional shortest-path algorithm, however it has been shown by Dubrovina et al. [44] that it cannot be used reliably to compute level line distance due to topological issues. It would yield a totally different set of level lines that do not represent correctly the image content. In fig. 41c, we show the level lines of the distance map using the Dijkstra shortest path algorithm. As one can see, it gives a totally different set of level lines and fails in retrieving the level line surrounding the plane.

6.5 SELECTING SHAPES BY PRIORITY

We have previously introduced operators that create new shapes by merging some marginal shapes. Consider now we want to restrict ourselves to existing marginal shapes and prevent the creation of new shapes. Rather we want to keep the largest subset $\mathcal{S}' \subset \mathcal{S}$ of the most "important" ones such that $(\mathcal{S}', \subseteq)$ forms a tree. The "importance" of a shape being given by a priority function $p : \mathcal{S} \to \mathbb{R}$, where a low value stands for a higher priority.

Consider the example in which we want to select every red shape, then every green shape that does not overlap a red shape and so on with the blue shapes.... The underlying priority function is:

$$p(A) = \min\{i \in \mathbb{N} \mid A \in \mathcal{S}_i\} \tag{6.4}$$

i.e., we give a priority "1" for shapes in the first tree, "2" for shapes in the second tree, and so on. Any other priority function can be used but it has to be contrast invariant if we want the final representation to be contrast invariant as well (any shape attribute would make the deal such as the elongation, the curvature, etc.).

Consider first $d : \mathcal{S} \to \mathbb{N}^n$ defined as:

$$\begin{aligned} d(A) = (&|\{X \in \mathcal{S}_1, A \subseteq X\}|, \\ &|\{X \in \mathcal{S}_2, A \subseteq X\}|, \\ &\dots, \\ &|\{X \in \mathcal{S}_n, A \subseteq X\}|) \end{aligned} \tag{6.5}$$

In other words, given the i^{th} tree, $d_i(A)$ is actually the number of shapes from it which each contain A.

Proposition 1. *Consider the product order $\preceq$ on $\mathbb{N}^n$ defined as:*

$$a \preceq b \Leftrightarrow \forall i, a_i \leq b_i \tag{6.6}$$

And the partial ordering relation $\prec_{\mathcal{S}}$ on $\mathcal{S}$ s.t.

$$A \preceq_d B \Leftrightarrow A \cap B \neq \emptyset \text{ and } d(A) \preceq d(B) \tag{6.7}$$

Then (1) d is an homomorphism from $(\mathcal{S}, \subseteq)$ to $(\mathbb{N}^n, \succeq)$ and (2) $(\mathcal{S}, \subseteq)$ is isomorphic to $(\mathcal{S}, \succeq_d)$.

Proof. 1. Let A and B be two shapes. If $A \subseteq B$, then any other shape containing B also contains A, thus $\forall i, d_i(A) \geq d_i(B)$ and: $A \subseteq B \Rightarrow d(A) \succeq d(B)$.
2. We need to proove that $A \subseteq B \Leftrightarrow A \succeq_d B$. The $\Rightarrow$ is straighforward by (1).
Now we proove that $A \succeq_d B \Rightarrow A \subseteq B$ by contradiction. Suppose $A \succeq_d B$ and $A \not\subseteq B$. We assume that $A \in \mathcal{S}_i$ and $B \in \mathcal{S}_j$. If $i = j$, A and B belongs to the same tree $\mathcal{T}_i$ thus A and B are either nested or disjoint. Since $A \cap B \neq \emptyset$ and $d_i(A) \geq d_i(B)$, it contradicts $A \not\subseteq B$. Consider now the case where $i \neq j$. We note $\mathcal{S}_j^A = \{X \in \mathcal{S}_j, A \subseteq X\}$. The shapes of $\mathcal{S}_j^A$ and $\mathcal{S}_j^B$ are either nested or disjoint. Since $A \cap B \neq \emptyset$ and $d_j(A) \geq d_j(B)$ then $\mathcal{S}_j^A \supseteq \mathcal{S}_j^B$. Since $B \in \mathcal{S}_j^B$ but $A \not\subseteq B$ then $B \notin \mathcal{S}_j^A$. That contradicts $\mathcal{S}_j^A \supseteq \mathcal{S}_j^B$ and finishes the proof. □

Let $\mathcal{S}_{\text{vlap}}^A$ denote the set of of shapes which overlap A:

$$\begin{aligned} \mathcal{S}_{\text{vlap}}^A &= \{X \in \mathcal{S} \mid A \cap X \neq \emptyset,\ X \not\subset A,\ A \not\subset X\} \\ &= \{X \in \mathcal{S} \mid A \cap X \neq \emptyset,\ d(X) \not\prec d(A),\ d(X) \not\succ d(A)\} \end{aligned}$$

We are actually interested in those shapes of lower priority. We denote $p'(A)$ the lowest priority of the shapes that overlap A:

$$p'(A) = \min_{X \in \mathcal{S}^A_{\text{vlap}}} p(X)$$

And finally we can define the shape selection by priority operator $\rho : \mathcal{S} \Rightarrow \mathbb{N}^n$:

$$\rho(A) = \begin{cases} d(A) & \text{if } p(A) \leq p'(A) \\ \bot & \text{otherwise} \end{cases}$$

Proposition 2. *For any two shapes $A \neq B \in \mathcal{S}$, either $A \cap B = \emptyset$ or $\rho(A) \preceq \rho(B)$ or $\rho(B) \preceq \rho(A)$*

Proof. Suppose $A \cap B \neq \emptyset$.
If $A \subset B$ or $B \subset A$ we have either $\bot \preceq d(A) \preceq d(B)$ or $\bot \preceq d(B) \preceq d(A)$ thus $\rho(A) \preceq \rho(B)$ or $\rho(A) \succeq \rho(B)$. Otherwise A and B overlap without being nested, thus $\rho(A) = \bot$ or $\rho(B) = \bot$ and then, $\rho(A) \preceq \rho(B)$ or $\rho(B) \preceq \rho(A)$ □

Proposition 3. *The maxtree of $\omega(x) = \overset{\curvearrowleft}{\max}_{X \in \mathcal{S}, x \in X} \rho(X)$ only contains the shapes of highest priority that do not overlap.*

Proof. Consider the set of shapes $\mathcal{S}' = \{A \in \mathcal{S} \mid \rho(A) \succ \bot\} \cup \{\Omega\}$. Then, $\forall A, B \in \mathcal{S}'$, $\rho(A) \preceq \rho(B) \Leftrightarrow d(A) \preceq d(B)$. We define the relation $\preceq_\rho$ on $\mathcal{S}'$ as in eq. (6.7) replacing d by ρ. It follows that $(\mathcal{S}', \succeq_\rho)$ is isomorphic to $(\mathcal{S}', \succeq_d)$ and by prop. 1, $(\mathcal{S}', \subseteq)$ is isomorphic to $(\mathcal{S}', \succeq_\rho)$. By prop. 2, it follows that $(\mathcal{S}', \subseteq)$ and $(\mathcal{S}', \succeq_\rho)$ form a tree. By prop. 5, the max-tree of ω is $(S', \subseteq)$. □

6.6 CONCLUSION

In this chapter, we have introduced a novel approach to extending the ToS to multivariate data. This novelty lies in the fact that we do not rely on any total ordering on values. Instead, we proposed merging the level lines issued from several ToS's computed on each component marginally. The way of merging those level lines is not dependant on their values but rather on their level of inclusion through a structure called the GoS. Instead of merging shapes, we have also introduced a way to select a set non-overlapping shapes given a priority criterion. In the next chapter, we will prove that trees generated by the proposed method meet some important criteria such as the marginal contrast change/inversion of contrast.

7

PROPERTIES OF THE MTOS

In chapter 6, we have introduced a method, the MToS, to extend the ToS to multivariate data. We also highlighted some properties of the ToS that we would extend to the MToS:

- the invariance to the change of constrast,
- the invariance to the inversion of contrast (self-duality),
- the support for connected operators.

A transformation ψ is said to be *contrast change invariant* if given a strictly increasing function $g : \mathbb{R} \to \mathbb{R}$, $g(\psi(u)) = \psi(g(u))$. Moreover, the transformation is said to be *self-dual* if it is invariant w.r.t. the complementation, *i.e.*, $\complement(\psi(u)) = \psi(\complement(u))$ (for images with scalar values $\complement(u) = -u$). When ψ is both self-dual and contrast change invariant, then for any strictly monotonic function G (*i.e.*, either strictly increasing or decreasing), we have $G(\psi(u)) = \psi(G(u))$. The ToS is actually a support for many self-dual morphological operators and a representation T is said to be self-dual and morphological if $T(G(u)) = T(u)$.

In this chapter, after a recall of topological considerations about the ToS in section 7.1, we are going to prove in section 7.2 that the method T, producing the MToS $T(\mathbf{u}) = (\boldsymbol{S}(\mathbf{u}), \subseteq)$, has the following properties:

(P0) *Well-formed tree* On "classical" images (not synthetic), the tree has a sufficient height and a sufficient number of nodes. In other words, it produces a tree which is topologically similar to the "classical" ToS.

(P1) *Domain covering* $\left(\bigcup_{X \in \boldsymbol{S}(\mathbf{u})} X\right) = \Omega$
(a point belongs to at least one shape)

(P2) *Tree structure*
$\forall X, Y \in \boldsymbol{S}(\mathbf{u})$, either $X \cap Y = \emptyset$ or $X \subseteq Y$ or $Y \subseteq X$
(any two shapes are either nested or disjoint)

(P3) If a shape $X \in \mathcal{S}$ verifies:

$$\forall Y \neq X \in \mathcal{S},\ X \cap Y = \emptyset \text{ or } X \subset Y \text{ or } Y \subset X$$

then $X \in \boldsymbol{S}(\mathbf{u})$ (any shape that does not overlap with any other shape exists in the final shape set). A corollary of this property is the *scalar ToS equivalence*. If $\mathcal{M} = \{\mathcal{S}_1\}$ then $\boldsymbol{S}(\mathbf{u}) = \mathcal{S}_1$, *i.e.*, for scalar images the tree built by the method is equivalent to the graylevel ToS.

(P4) *Marginal contrast change/inversion invariance.*
Let us consider $\mathbf{G}(\mathbf{u}) = (G_1(u_1), G_2(u_2), \ldots, G_n(u_n))$, where G_i is a strictly monotonic function, then T is invariant by marginal inversion/change of contrast; that is, $T(\mathbf{G}(\mathbf{u})) = T(\mathbf{u})$.

7.1 TOPOLOGICAL CONSIDERATIONS

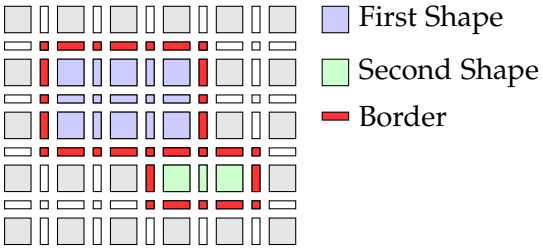


Figure 42: Shapes on the cubical grid (here the 2D square grid).

Throughout this paper, we assume that the image has its domain on a cubical grid that allows continuous properties while staying on a discrete space. The algorithm proposed by Géraud et al. [46] to compute the ToS in grayscale uses this representation as well and more advanced details about topological properties with this grid can be found in [94]. We simply recall basic notions that will be necessary for the proofs in this chapter. We denote $\mathcal{K}_\Omega$ the domain Ω immersed on the cubical 2D grid. Previously, as a matter of clarity, we have denoted Ω the domain of the image, but $\mathcal{K}_\Omega$ was always assumed. In fig. 42, original pixels are represented by 2-faces (large square) and intermediate pixels are added (1-faces and 0-faces). In the ToS, a *shape* A is an open set on the grid and may be composed of 0, 1 and 2-faces. In fig. 42, red and green elements represent two disjoint shapes A and B. On the other hand, the border of a shape is composed of 0 and 1-faces only (red elements). Shape boundaries are the actual level lines of the image. We denote ∂A, the border of the set A and $\bar{A} = A \cup \partial A$ the closure of A. Note that in the tree of shapes, two shapes are either nested or disjoint but $\bar{A}$ and $\bar{B}$ may overlap as shown in fig. 42.

Proposition 4. *Let a shape $A \in \mathcal{S}$, and $x \in \partial A$, then* $\forall\, X \in \mathcal{S},\ x \in X \ \Rightarrow\ X \in A\uparrow$.

Proof. Suppose $x \in \partial A$, and a shape $B \in \mathcal{S}$ such that $x \in B$. Then, B is an open set, so it contains the 2-face in A adjacent to x and $B \cap A \neq \emptyset$. Two shapes being either disjoint or nested, we have $B \subseteq A$ or $A \subseteq B$. Since $x \in B$ but $x \notin A$, it follows that $A \subseteq B$ and $B \in A\uparrow$. □

7.2 PROOF OF CORRECTNESS

We now prove that the MToS construction process meets the properties given at the beginning of the chapter.

Proposition 5. *Given a set of shapes $\mathcal{S}$ where any two shapes are either disjoint or nested ($(\mathcal{S}, \subseteq)$ is a tree) and a strictly decreasing attribute ρ then $\mathbb{C} = \mathcal{S}$*

Proof. ($\Rightarrow$) Let a shape $A \in \mathcal{S}$. $\forall x \in A$, we have:

$$\omega(x) = (\max_{X \in \mathcal{S}, x \in X} \rho(X)) \geq \rho(A)$$

Let now, $x \in \partial A$, $\forall X \in \mathcal{S}, x \in X$, we have $A \subset X$ (by prop. 4), thus $\rho(X) < \rho(A)$ and $\omega(x) < \rho(A)$. It follows that $A \in \mathcal{CC}([\omega \geq \rho(A)])$ and $A \in \mathbb{C}$.
($\Leftarrow$) Let $A \in \mathcal{P}(\Omega)$, $A \notin \mathcal{S}$, we note $\mathrm{SES}(A)$ the smallest enclosing shape that includes A. Suppose now that $A \in \mathbb{C}$, then $\exists \lambda \in \mathbb{R}, A \in \mathcal{CC}([\omega \geq \lambda])$. Let $\alpha \in \mathbb{R}$, $\alpha = \min_{x \in A} \omega(x)$, then $\lambda \leq \alpha$. Yet,

$$\begin{aligned}
\alpha = \min_{x \in A} \omega(x) &= \min_{x \in A} \max_{X \in \mathcal{S},\, x \in X} \rho(X) \\
&= \min_{X \in \mathcal{S},\, A \cap X \neq \emptyset} \rho(X) \\
&= \rho(\mathrm{SES}(A))
\end{aligned}$$

Thus, $A \in \mathcal{CC}([\omega \geq \lambda])$ with $\lambda \leq \rho(\mathrm{SES}(A))$. But, since $A \subsetneq \mathrm{SES}(A)$ and $A, \mathrm{SES}(A) \in \mathbb{C}$ then $\lambda > \rho(\mathrm{SES}(A))$ which contradicts $\lambda \leq \rho(\mathrm{SES}(A))$. Therefore, $A \notin \mathcal{S} \Rightarrow A \notin \mathbb{C}$ □

A direct consequence of prop. 5 is that given a ToS $\mathcal{T}$ and the max-tree $\mathcal{T}_\omega$ of the image ω reconstructed from a decreasing attribute ρ over $\mathcal{T}$, then $\mathcal{T}_\omega = \mathcal{T}$.

Proposition 6. *The method provides a tree $\mathcal{T}_\omega$ that verifies the property (P1), (P2), (P3), and property (P4) if ρ is a pure algebraic attribute (it does not depend on the values of u).*

Proof. (P1) and (P2) are straightforward because the 5^{th} step consists in computing a max-tree with hole-filled components on a scalar image ω. The shapes of the hole-filled maxtree being a subset of the shapes of the ToS of ω, it follows that any two shapes are either nested or disjoint.
(P3) With the same proof as in prop. 5, we show that $\forall A \in \mathcal{S}$, if $\forall B \in \mathcal{S}$, $A \cap B = \emptyset$ or $A \subseteq B$ or $B \subseteq A$, then $A \in \mathbb{C}$ and $A \in \mathbf{S}$.
(P4) A marginal tree $\mathcal{T}_i$ only depends on the i^{th} channel, thus it is invariant w.r.t. to u_j ($j \neq i$). By property of the ToS, $\mathcal{T}_i$ is contrast change/inversion invariant w.r.t. u_i. It follows that every $\mathcal{T}_i$ is marginally invariant w.r.t. u and so does the set of shapes $\mathcal{S}$ to build the graph $\mathfrak{G}$. Since the rest of the process only depends on the graph topology and no more on the values of u, $\mathcal{T}_\omega$ is thus marginally contrast change/inversion invariant. □

Proposition 7. *The method provides a tree $\mathcal{T}_\omega$ which is well-formed (Po).*

Image	Size	#nodes	Avg. depth	Max depth
airplane	262k	81k / 129k	75 / 78	234 / 197
baboon	240k	89k / 129k	41 / 63	95 / 127
barbara	414k	141k / 228k	64 / 136	203 / 306
boats	453k	119k / 208k	83 / 163	232 / 295
goldhill	414k	123k / 240k	58 / 105	223 / 295
house	65k	22k / 35k	36 / 77	154 / 175
lenna	262k	69k / 161k	40 / 59	148 / 193
pepper	262k	102k / 200k	43 / 117	148 / 291

Table 1: Tree statistics comparison on well-known test images between the ToS on the gray-level image (left side of the columns) and the MToS on the color image (right side of the columns).

To verify the property (Po), we have computed some statistics about the tree topology on some classical images. In table 1, we show the number of nodes, the average node depth and the height of the MToS compared to the ToS computed on the gray level version of the image. The number of nodes in the MToS is 50% to 100% higher than in the ToS that highlights a better precision. Moreover, the average depth of the nodes (as well as the height of the tree) increases significantly meaning that we do not just add some leaf nodes (noise) but rather, some large shapes that increase the shape inclusion chains.

7.3 CONCLUSION

In this chapter, we have proven that the process exposed in chapter 6 builds the MToS which statisfies the desired properties; that is: a hierarchical structure, organizing hole-free connected components in terms of inclusion, which is invariant to any marginal change or inversion of contrast. We have seen that those properties are of main interest in a context requiring a certain robustness to a change of illumination or a change of scene view since these properties should ensure that the topology of the tree remain globally the same in those cases. Yet, we have not discussed the properties of reconstruction from the MToS. Since the image cannot be recovered from the MToS, one can wonder which restitution strategy would ensure a *idempotent* filtering? Indeed, we have seen in section 3.2, that depending on the filtering strategy, the reconstructed image may not correspond to the tree from which the reconstruction was based. Here, the problem is even harder. Since many values might be associated with a single node, we have to construct a rule for selecting the filtering values when nodes are removed. The questions about the properties of the restitution are thus postponed as a further work.

8

MTOS COMPUTATION ALGORITHM

In chapter 6, we have introduced a two-step process to compute the MToS. The first step requires the computation a new structure called the Graph of Shapes (GoS). The second consists of computing a tree from the graph. In this chapter, we will introduce the algorithms involved in those two steps in section 8.1 and section 8.2. Eventually, in section 8.3, we will discuss the complexity of those algorithms.

8.1 COMPUTING THE GRAPH OF SHAPES

To start with, we have to compute the ToS's on each channel independently. We rely on the algorithm of Géraud et al. [46] and we use the same tree encoding as they do. That is, a ToS is encoded through a *parent* image where the parent relationship denotes the inclusion of shapes. A shape is actually represented by a single point, namely the *canonical element*. This point serves as the *parent* for all other points of the shape. We suppose the function getCanonical(x) that returns the canonical element of the node that contains x.

In the following, lowercase symbols (e.g. $x, y, p, \ldots$) denote points in Ω while uppercase symbols (e.g. A, $B \ldots$) denote points that are canonical elements. By abuse of notation, we let A denote three things: the shape (element of $\mathcal{P}(\Omega)$), the tree node (the elements of A that are not in a sub-shape) and the canonical element (element of Ω) that represents A.

A straightforward algorithm to compute $\mathfrak{G}$ would be to test inclusion of shape of one tree with every shape of the other tree to get the graph of inclusion and then perform its transitive reduction. It would lead to a complexity of $O(n^3)$. This complexity can drop to n^2 by pre-computing the Smallest Enclosing Shape (SES) of every shape in every tree using dynamic programming that enables removing the transitive reduction of the final graph.

Finding the Smallest Enclosing Shape of A in a shape set $\mathcal{S}$, *i.e.*,

$$SES(A) = \overset{\subseteq}{\bigwedge}\{X \in \mathcal{S}, A \subseteq X\}$$

can be formulated as a well-known Least Common Ancestor problem on lattices. Indeed, the smallest enclosing shape of A is the least common ancestor of all $p \in A$. Moreover, since $SES(A \cup B) = LCA(SES(A), SES(B))$, a dynamic programming formulation of the problem exists and enables a fast recursive computation on trees:

$$SES(A) = LCA(SES(C_1), \ldots, SES(C_k), x_1, \ldots, x_{k'}) \tag{8.1}$$

Algorithm 1: Least Common Ancestor

Input: A, B: two shapes in $\mathcal{S}_i$
Output: The least common ancestor of A, B in $\mathcal{T}_i$
Data: *par* The parent image encoding $\mathcal{T}_i$
depth The depth attribute on $\mathcal{T}_i$

if $A = NULL$ **then return** B;
if $B = NULL$ **then return** A;
while $depth(A) < depth(B)$ **do** $B \leftarrow par(B)$;
while $depth(B) < depth(A)$ **do** $A \leftarrow par(A)$;
while $A \neq B$ **do** $A \leftarrow par(A)$; $B \leftarrow par(B)$;
return A;

Algorithm 2: Smallest Enclosing Shape

Data: *par* The parent image encoding $\mathcal{T}_i$
Output: The SES attribute for $\mathcal{T}_i$ w.r.t $\mathcal{T}_j$

$SES(A) \leftarrow NULL$ foreach $A \in \mathcal{T}_i$;
foreach *point* $x \in \Omega$ **do**
  $A \leftarrow$ getCanonical(x) in $\mathcal{T}_i$;
  $X \leftarrow$ getCanonical(x) in $\mathcal{T}_j$;
  $SES(A) \leftarrow \text{LCA}(SES(A), X)$ in $\mathcal{T}_j$;
foreach *node* $A \in \mathcal{T}_i$ *upward* **do**
  $Q \leftarrow par(A)$;
  $SES(Q) \leftarrow \text{LCA}(SES(Q), SES(A))$ in $\mathcal{T}_j$;
return *SES*

where $C_1, \ldots, C_k$ are the children of A and $x_1, \ldots, x_{k'}$ are the points contained in the node A. Algorithm 1 computes the least common ancestor of two shapes A and B by following the paths of A and B up to the root and stops as soon as the paths join each other. This relies on the *depth* of nodes to advance the path at the same tree level so the depth attribute has to be computed as pre-processing. Algorithm 2 computes for each shape A of the i^{th} tree its smallest enclosing shape in the j^{th} tree. In the following, we denote this smallest enclosing shape by $SES_{ij}(A)$. It is a straightforward attribute computation that processes the tree in a bottom-up fashion using eq. (8.1).

To compute the GoS, we rely on the output of the smallest enclosing shape algorithm to get a graph which is almost already reduced. Algorithm 3 proceeds as follows. It first adds new vertex in the graph for each shape of the trees $\mathcal{T}_1, \ldots, \mathcal{T}_d$, taking care of not to add a shape twice if it belongs to several trees. For that purpose, we rely on the smallest enclosing shape attribute computed previously. Indeed, if two shapes X_i and X_j from respective trees $\mathcal{T}_i$ and $\mathcal{T}_j$ are equal, then we have $\text{SES}_{ji}(X_j) = X_i$ and $\text{SES}_{ij}(X_i) = X_j$.

Algorithm 3: Graph of shapes computation algorithm

Output: $\mathfrak{G} = (V, E)$ the graph of shapes, with V the set of shapes and E the set of edges
Data: $\mathcal{T}_i = (\mathcal{S}_i, \subseteq)$ The marginal tree of shapes
Data: par_i The parent image encoding $\mathcal{T}_i$
Data: SES_{ij} The SES attribute on $\mathcal{T}_i$ w.r.t $\mathcal{T}_j$
Data: d The graph attribute as defined in eq. (6.5)

```
1  V ← ∅; E ← ∅;
2  { add nodes to V };
3  for i ← 1 to d do
4      foreach node A in T_i do
5          skip ← False;
6          for j ← 1 to i − 1 do
7              if SES_ji(SES_ij(A)) = A then                    // A ∉ V
8                  skip ← True;
9          if not skip then V ← V ∪ {A} ;
10 { add edges to E };
11 foreach tree T_i do
12     foreach node A in T_i do
13         E ← E ∪ {(A, par_i(A))};
14         foreach j ← 1 to d such that A ∉ T_j do
15             E ← E ∪ {(A, SES_ij(A))};
16 { remove from E all the edges that are not in the cover };
17 foreach vertex A ∈ V do
18     foreach pair of edges (A, U), (A, U') do
19         if d(U) ≻ d(U') then E ← E \ (A, U');
20 return 𝔊 = (V, E);
```

Then, the condition $\mathrm{SES}_{ji}(\mathrm{SES}_{ij}(X_i)) = X_i$ at line 7 with $j < i$ allows to detect if the shape has already been inserted before.

Next, the edges of the graph are added. Each shape is linked to its parent and to every smallest enclosing shape in the other trees. This way, we prevent inserting a lot of edges that would otherwise be removed during the reduction step. However, some edges still need to be removed. Consider for example the simple case where we have three shapes A, B, C (one in each tree) such that $A \subset B \subset C$. Then, the graph would contain the edges (A, B), (A, C) and (B, C) but (A, C) is not in the cover of the graph. Thus, the last step of the algorithm aims at removing $\{(A, C) \mid \exists B,\ A \subset B \subset C\}$. An efficient way to know if two intersecting shapes are nested is to rely on the attribute $d(A) =$

$(depth_1(SES_1(A)), \ldots, depth_n(SES_n(A)))$ defined in section 6.5. We know by prop. 1 that $(\mathcal{S}, \subseteq)$ is isomorphic to $(\mathcal{S}, \succeq_d)$ thus $\forall B, C \in \mathcal{S}, B \cap C \neq \emptyset$ and $d(B) \succ d(C) \Rightarrow B \subset C$.

8.2 TREE EXTRACTION ALGORITHM

8.2.1 *Computing the distance map*

We have seen in section 6.4 that the second part of the method involves computing an attribute over the graph and constructing a *distance* map that renders the level of inclusion of each pixel. We have proposed three *depth* functions that we succinctly reiterate here:

d_1: $\omega_{\mathrm{CV}}(x) = |X \in \mathcal{S}, x \in X| - 1$
Counts the minimal number of marginal level lines we need to traverse to get x from the border.

d_2: $\rho(A) = |A\uparrow|$ and $\omega_{\mathrm{CV}}(x) = \max_{X \in \mathcal{S},\, x \in X} \rho(X)$
Counts the number of marginal level lines to traverse to get the deepest shape that contains x.

d_3: $\rho(A) = \max_{\phi \in [\Omega \rightsquigarrow A)} |\phi|$ and $\omega_{\mathrm{CV}}(x) = \max_{X \in \mathcal{S},\, x \in X} \rho(X)$
where $[\Omega \rightsquigarrow A)$ stands for the set of paths from the root to A in $\mathfrak{G}$. It counts the number of *nested* marginal level lines to traverse to get the deepest shape that contains x.

As said in section 6.4, D_3 is straightforward to compute as one just has to apply a shortest path algorithm from the root. However, it requires a small modification as we want for each node, the length of the longest path instead of the shortest.

On the other hand, computing D_1 and D_2 are a bit more challenging as we have to avoid counting the nodes twice during the accumulation of values along the paths. For that purpose, we are going to rely on individual trees as there exists a single path from the root to any node, thus easing the process. Let,

$$\mathrm{Uniq}_i(X) = \begin{cases} 0 & \text{if } \exists\, 1 \leq j < i,\ \mathrm{SES}_{ji}(\mathrm{SES}_{ij}(X)) = X \\ 1 & \text{otherwise} \end{cases}$$

The function $\mathrm{Uniq}_i(X)$ tells for each shape X of $\mathcal{T}_i$ if it is already present in a tree $\mathcal{T}_j$, $j < i$. If so, the shape counts for zero, one otherwise. As for the algorithm 3, the test $\mathrm{SES}_{ji}(\mathrm{SES}_{ij}(X)) = X$ allows us to know whether a shape of $\mathcal{T}_i$ belongs also to $\mathcal{T}_j$. Now, we can define the function $\mathrm{d'}_i(A)$ that computes the number of shapes in $\mathcal{S}_i$ that include A but are not available in any other tree $\mathcal{T}_j$, $j < i$:

$$\mathrm{d'}_i(A) = \sum_{X \in \mathcal{S}_i,\, A \subseteq X} \mathrm{Uniq}_i(X)$$

Algorithm 4: Computation of the depth without shape redundancy.

Output: d'_i The depth attribute without shape redundancy.
Data: $\mathcal{T}_i = (\mathcal{S}_i, \subseteq)$ The marginal tree of shapes
Data: par_i The parent image encoding $\mathcal{T}_i$
Data: SES_{ij} The SES attribute on $\mathcal{T}_i$ w.r.t $\mathcal{T}_j$

```
foreach node A in T_i downward do
    uniq ← True;
    for j = 1 to i do
        if SES_ji(SES_ij(A)) = A then
            uniq ← False;
    if uniq then
        d'_i(A) ← d'_i(par_i(A)) + 1;
    else
        d'_i(A) ← d'_i(par_i(A));
return d'_i;
```

Algorithmically speaking, this is a minor change to *depth* attribute computation algorithm. This is illustrated in algorithm 4, where for each node A of $\mathcal{T}_i$, we first check if it does exist in any previous tree $\mathcal{T}_j$ and, according to the result, we then increment the depth from its parent's value.

Let S_i^x denote the smallest shape containing x in $\mathcal{S}_i$, and $S^x = \{ S_1^x, \ldots, S_d^x \}$, then:

- D_1 involves computing the number of shapes containing x which is also the number of inclusions of S_i^x in each tree minus the size of the intersection of the shape sets. So:

$$\omega_{\mathrm{CV}}(x) = \sum_{i=1}^{d} \mathrm{d}'_i(S_i^x)$$

- D_2 involves computing the number of ancestors of A in the graph. This is actually the number of inclusions marginally in each tree minus the number of shapes that are common to these trees.

$$\rho(A) = \sum_{i=1}^{d} \mathrm{d}'_i(A)$$
$$\omega_{\mathrm{CV}}(x) = \max_{X \in S^x} \rho(X)$$

8.2.2 *Computing the hole-filled max-tree*

Once the distance map ω is built, we need to compute its hole-filled max-tree. The straightforward algorithm that consists of computing the max-tree, filling its components

and testing their inclusion to build back a new tree would lead to a high complexity. Since the shapes of the ToS are actually a super-set of the hole-filled components of the max-tree, a better method is to compute the ToS and filter out shapes issued from lower threshold sets. We have seen in section 3.2.3, that the direct filtering cannot be used reliably as the reconstructed image may no match the filtered tree. So we have to use a subtracting filtering ensuring that the value of the nodes in the new tree are strictly increasing when traversing it downward.

8.3 COMPLEXITY ANALYSIS

Let $N = |\Omega|$ denote the number of pixels in the original image and d be the dimension of F. As pre-processing, we have to compute the ToS's marginally on each channel and the depth attribute. A single tree computation is quasi-linear [1] ($O(N \log\log N)$ for high-quantized data) and a simple attribute computation is $O(N)$ (as there are at most as many nodes in the tree as the number of pixels). Thus, the pre-processing is $O(d.N.\alpha N)$. Algorithm 2 requires for each point a least common ancestor computation which is $O(H)$, where H is the depth of the ToS so $O(N.H)$. In the degenerated case, where the tree is a chain, we have $H = N$, but in practice the depth of the trees is much less than N, so we use the average complexity H (on natural images, $H \simeq 300$). Algorithm 3 requires algorithm 2 to be computed on each pair of tree ($O(d^2.N.H)$), next adds the nodes to V ($O(d^2.N)$), then adds the edges to E ($O(d^2.N)$) and finally reduces the graph ($O(d^2N)$) (because $|V|$ is at most $d.N$, and each node has at most d outer edges). Finally, the overall complexity of the process is dominated by the computation of SES_{ij}; that is $O(d^2.H.N)$. However, the parallelization of this computation is straightforward since the SES_{ij} are not dependent on each other.

Let us now focus on the second part of the method, where we have to build the distance map ω. Using the distance D_3, we have to compute the longest path length from the root to any node. A Dijkstra-like algorithm is $O(|E| + |V| \log |V|)$, but since the graph is acyclic, we can use a topological sorting to calculate the root source longest distances in $O(|E| + |V|) = O(d.N)$. The other distances D_1 and D_2 use a single pass algorithm over each trees so $O(d.N)$ as well. Finally, to get the final hole-filled max-tree, we have to build the ToS and perform a simple filtering. The complexity is dominated by the construction process, which is $O(N.\alpha(N))$.

8.4 CONCLUSION

In this chapter, we have given the algorithms involved in the computation of the MToS. The process has two main parts: the construction of the GoS and the computation of a tree from it. We have seen that the most computationally expensive part is the construction of the graph, as it requires to retrieve for any shape of each tree, its smallest enclosing shape

1 $O(N.\alpha(N)$ where $\alpha(n)$ is the inverse of the Ackermann function which increases slowly.

in any other tree. The second part being just an attribute computation over the graph to get the distance map and its ToS computation; it is negligible compared to the first part. The whole process is quasi-linear in the number of pixels, but quadratic in the number of dimension, that makes it, at first glance non-practicable for high dimensional data. However, we have noted that the parallelization of the quadratic part is straightforward as it is a process on each marginal tree.

9 EMPIRICAL VALIDATION OF THE MTOS

In chapter 6 and chapter 7, we have introduced a new hierarchical representation of multivariate images which meets important properties regarding the invariance to any marginal change or inversion of contrast. This approach, that extends the ToS to multivariate data, tries to merge some marginal shapes in a sensitive way, by relying on the inclusion relationship between marginal components only. As a consequence, one may wonder the influence of some perturbations on one channel in the merging process. This chapter is two-fold. In section 9.1, we compare the performance of the MToS to the most standard approaches exposed in chapter 4 and show practically that it is more accurate and avoids most artifacts present with the other approaches. Then, in section 9.2, we lead experiments to study the influence of merging with a noisy components and with components of different dynamics and show its robustness to both type of perturbations.

9.1 COMPARISON WITH STANDARD APPROACHES

We have seen in chapter 4, some classical methods to extend the ToS to multivariate data. In this section, We review some of them and compare their performance with our MToS.

An unacceptable but widely used workaround for color image processing is to get rid of the colors and process a gray-level version of the multivariate image. This workaround makes sense if we pretend that the geometric information is mainly held by the luminance [31]. However, it is not that rare to face images where edges are only available in the color space (especially document and synthetic images). They contradict this assumption and prove that the chrominance holds the geometric information as well, as shown in fig. 43b where the luminance is not sufficient to retrieve the whole geometric content.

Another commonly used solution is processing the image channel-wise and finally recombine the results. Marginal processing is subject to the well-known false color problem as it may create new values that were not in the original images. False colors may or may not be a problem in itself (*e.g.* if the false colors are perceptually close to the original ones) but for image simplification it may produce undesirable artifacts as shown in fig. 43c.

Since the pitfall of overlapping shapes is due to the partial ordering of multivariate data, some authors tend to impose an "arbitrary" total ordering or total (pre)ordering on values. The most advanced strategies have been designed to build a more "sensitive" total ordering that depends on the image content. We have seen for example that in [74], manifold learning is used to infer a ranking function of values and in [72] a locally-dependent ordering are computed on spacial windows. [71] combines both ideas for a manifold learning in a domain-value space capturing small dependencies between a pixel

(a) Original image u.

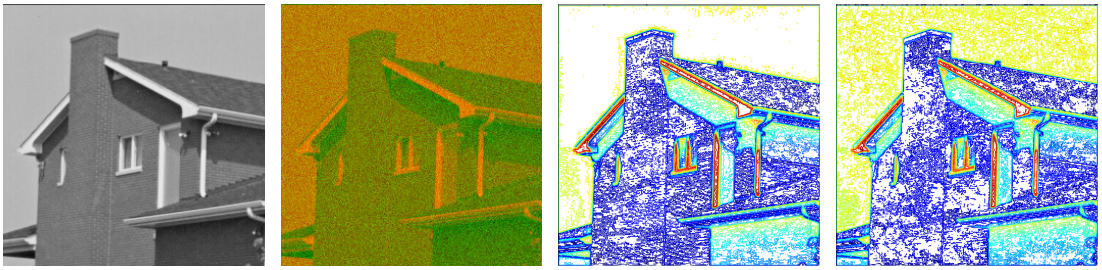
(b) Simplification on a gray-level version of u (198 regions).

(c) Simplification with a marginal processing (123 + 141 + 136 regions).

(d) Simplification with a nonlocal rank transformation [71] (total order) (171 regions).

(e) Simplification with our approach (158 regions).

Figure 43: Simplification issues with "standard" color image processing. (b) and (d) show leakage problems with total ordering. (c) illustrates the false color problem due to marginal processing. (e) shows that the MToS retrieves correctly the main content of the image while preventing false colors.



(a) (b) (c) (d)

Figure 44: Effects of merging with a noisy component. (a) Original image. (b) House (red channel) + Gaussian Noise ($\sigma = 20$, green channel) (c) Level lines of the ToS of (a). Level lines: 24k, avg. depth: 37, max. depth: 124. (d) Level lines of the MToS of (b). Level lines: 48k, avg. depth: 48, max. depth: 127.

and its neighbors during the construction of the total order. This is illustrated on fig. 43d where we have better results than using the luminance only but still have leakage issues.

On this example, as shown in fig. 43e, the MToS we have proposed retrieves correctly the main content of the image while preventing false colors.

9.2 ROBUSTNESS TO DYNAMICS AND NOISE

9.2.1 *Rubustness to noise*

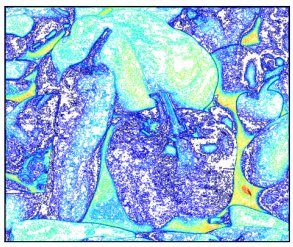
In fig. 44, we have conducted an experiment showing the robustness of the MToS to the noise. A 2-channel image has been built from a gray-level image for the first component and a random gaussian noise for the second one. In fig. 44d, we show that the presence of noise in the second channel does not alter the geometric information provided by the first channel as the meaningful level lines in fig. 44d match the ones in fig. 44c. The noise add new level lines at leaves level.

9.2.2 *Robustness to dynamics*

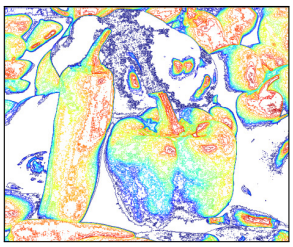
In fig. 45, the experiment shows the effect of the dynamics on the ToS. From the "pepper" image, we have extracted the two first components and black out the third (blue) one (fig. 45a). We have then limited the quantum of the green channel to 10 levels (fig. 45b). Figures 45c to 45e show the level lines of the red channel, the green channel, and the sub-quantified green channel respectively. Eventually, we have computed the MToS's of (a) and (b) to study the effect of merging components of equal and different dynamics; the corresponding level lines are shown in figs. 45e and 45g. If the components have similar dynamics, we retrieve shapes from both channels. Indeed, we see in fig. 45e, shapes coming from both (c) and (d). On the other hand, in the presence of an high-

(a) Peppers (only red/green channels)

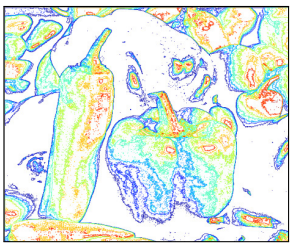
(b) Peppers (only red/green channels) with the green channel sub-quantized to 10 levels



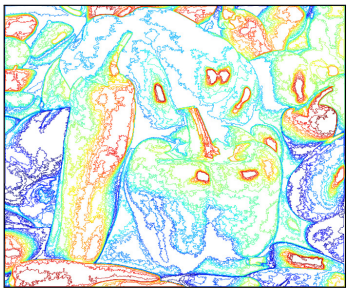
(c) Level lines of the red channel of (a) and (b)



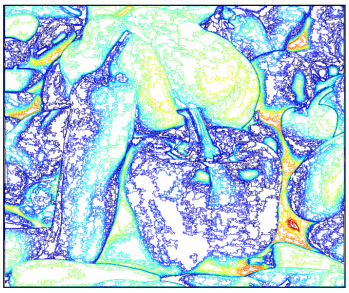
(d) Level lines of the green channel of (a)



(e) Level lines of the green channel of (b)



(f) Level lines of the MToS of (a)



(g) Level lines of the MToS of (b)

Figure 45: Effects of merging with a low-dynamic component.

dynamic channel and a low-dynamic channel, the MToS retains mainly the level lines of the high-dynamic one. Shapes in (g) are mainly coming from (c) and the ones of (e) are ignored. This also highlights that a component with fewer information does not disturb the MToS in retrieving the shapes from the one with more information.

9.2.3 *Merging unrelated geometric information*

The last experiment we have conducted is about merging unrelated geometrical information from different sources. In fig. 46, we have combined two gray-level images into a single 2-channel one on which we computed the MToS. The two main objects of the pictures are retrieved in the MToS which contains the shapes of both the butter and

Figure 46: Merging unrelated geometric information. The two first gray-level pictures (butter and fly) are merged into a single 2-channel images whose level lines given by the MToS are depicted in the third picture.

the fly. It shows the ability to our approach to merge information split across different channels of the image.

9.3 CONCLUSION

In this chapter, we have led some experimentation showing the pertinence and the accuracy of the MToS compared to standard approaches that impose a total ordering on values. We have seen practically that it can avoid leakage effects or color artifacts that are present with the other methods. We have also shown the robustness of the MToS to noisy channels or low-dynamic channels, which is of prime importance since it means that a channel which brings no information does not prevent our method to retrieve sensitive information from the other channels.

Part III

APPLICATIONS

10 IMAGE FILTERING

10.1 GRAIN FILTERS

A grain filter [29, 108] is an operator that removes the regions of the image which are local extrema and whose area is below a given threshold. In that sense, it is related to extreme filters but ensures a symmetric processing of the minima and the maxima, *i.e.*, it is self-dual. Using the ToS, a grain filter is thus a simple pruning, removing the nodes which do not pass the size criterion. This is illustrated in fig. 47. We assign for each node the size of the components and remove those nodes whose size is below 4, *i.e.*, every node below the red dashed curve. The pixels they contain are then attached to the first ancestor above the red curve. Despite the simplificy of this filter, we will see its power for image simplification and document layout extraction in section 10.1.1 and section 10.1.2 respectively.

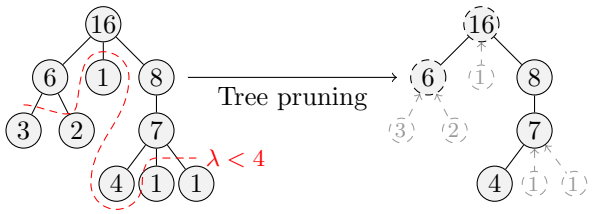
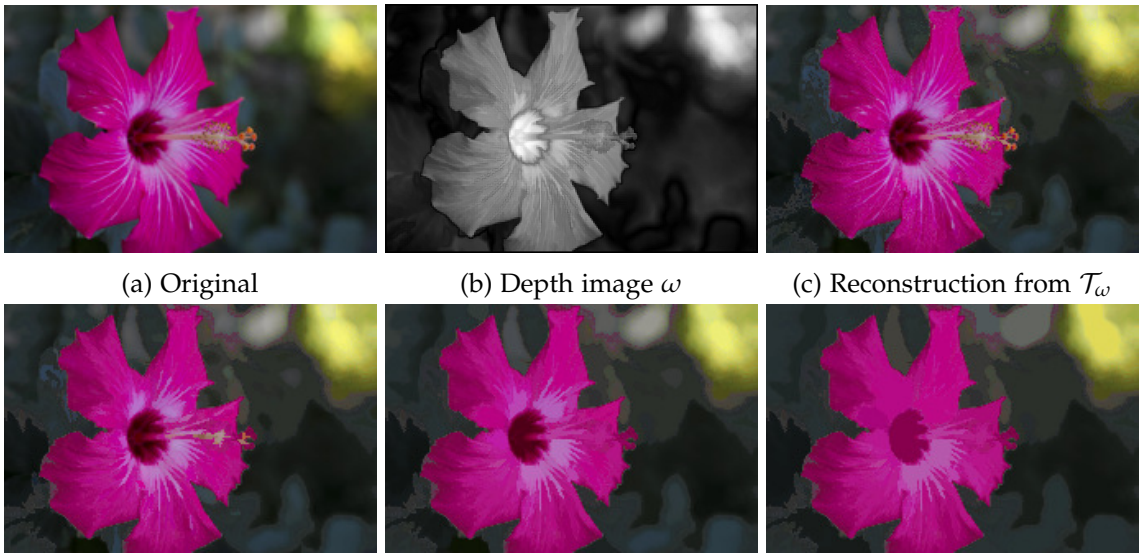


Figure 47: Scheme of a grain filtering process.

10.1.1 *Image simplification*

Grain filters allow us to reveal the "correctness" of the tree in the sense that a small grain size should filter out what we perceive as noise or details while an high grain size should show the main objects and the structure of the image. In fig. 48, we show the inclusion map ω computed using our method and the image reconstructed from the max-tree $\mathcal{T}_\omega$. The reconstruction consists of computing for each node the average color of the pixels it contains and then, assigning this value to the pixels, this is the $\mathrm{P}_{\mathrm{mean}}$ strategy seen in section 6.3.1. Because $\mathcal{T}_\omega$ is not a reversible representation of $\mathbf{u}$, the latter cannot be recovered from $\mathcal{T}_\omega$, however the reconstruction is close to the original. In fig. 48d, we have applied size-increasing grain filters that eliminate details in a "sensitive" way and provide a reconstruction with few color artifacts that validate the structure organization of our tree.



(a) Original (b) Depth image ω (c) Reconstruction from $\mathcal{T}_\omega$

(d) Grain filters of sizes 10, 100 and 500

Figure 48: Grain filters

10.1.2 *Document layout extraction*

We use a grain filter to extract text boxes and graphical parts of documents. Indeed, text parts are composed of letters which are supposed to be small components if the MToS is well-formed. On the contrary, text boxes and graphical contents are large components that should remain after the filtering. Figure 49 shows the extraction of non-textual content where self-duality matters since text may be over a darker or brighter background. As one can see, the filtered images only contain the graphical content and text boxes while actual letters are in the residue.

10.2 ENERGY MINIMIZATION CONSTRAINED TO TREES' TOPOLOGY

In [32], authors claim that the significant contours of objects actually correspond to some segment of the level lines of the image. As a consequence, image simplification or segmentation can be interpreted in terms of a selection of some *meaningful* level lines in the ToS [23, 21]. Following the same idea, [13, 101], and later [142], proposed simplification methods that try to select the subset of level lines such that the reconstructed image minimizes the simplified Mumford-Shah cartoon model [87]. This is so an energy optimization constrained by the tree topology. More formally, we have to select a subset of shapes $\mathcal{S}' \subset \mathcal{S}$ that minimizes:

$$E(\mathcal{S}') = \sum_{S \in \mathcal{S}'} \sum_{x \in S | S_x = S} ||u(x) - \bar{u}(S)||_2^2 + \lambda \, |\partial S|, \tag{10.1}$$

Figure 49: Simple filtering for document layout detection. Top row: original images; bottom row: results of grain filters.

where S_x denotes the smallest shape containing x, $\bar{u}(S)$ is the average color of the region and $|\partial S|$ the length of the shape boundary. In [13], the authors proposed a greedy algorithm that iteratively removes shapes from $\mathcal{S}$. It is composed of the following steps:

1. Initialize $\mathcal{S}' = \mathcal{S}$
2. For each node (shape) S of $\mathcal{S}'$, compute $\Delta E(S)$ which tells how much the energy decreases, *i.e.*, $\Delta E(S) = E(\mathcal{S}') - E(\mathcal{S}' \setminus \{S\})$
3. Search for the optimal shape $S^* = \arg\max_{S \in \mathcal{S}'} \Delta E(S)$ (we note $\text{par}(S^*)$ its parent) and remove S^* from $\mathcal{S}'$.

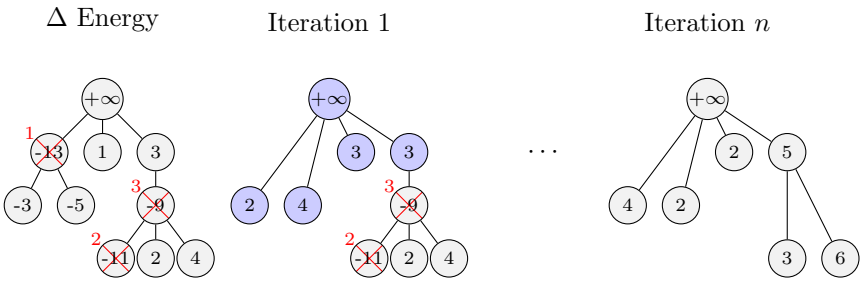


Figure 50: Scheme of the algorithm for energy minimization on trees. Left: the nodes are valuated with ΔE, red labels stand for their meaningfulness priority. Middle: at the first iteration, node 1 is removed, the blue nodes have their energy updated. Right: after several iteration, all ΔE are positive; we cannot remove any that would decrease the global energy; we have reached a local minimum.

4. Update the energy ΔE of $\text{par}(S^*)$ and its children. The energies of those shapes are the only ones to be affected by the removal. These nodes actually correspond to the parent, the children, and the siblings of the old shape S^*.

5. Go back to (3) until reaching stability.

At the end of the process, we reach a local optimum in the sense that no more shapes can be removed without increasing the energy. From an implementation point of view, the method requires maintaining a priority queue of the nodes w.r.t. their energy. In [142], the authors proposed replacing that priority queue by a predefined processing ordering of the nodes to speed up the process. Typically, the nodes are first sorted w.r.t. their meaningfulness (e.g. the magnitude of gradient on boundaries). This avoids updating the heap used for the priority queue and drops the complexity from $O(N \log N)$ to linear time. An example of such a process is illustrated in fig. 50.

The steps (2) and (4) of the method requires us to update the costs ΔE, so this computation must be effective to ensure a fast processing. Let X be a shape of $\mathcal{S}'$, $Y = \text{par}(X)$ its parent, and X_{pr}, Y_{pr} the "proper" pixels of the shapes (*i.e.*, the pixels that are not in any descendants). Then, ΔE can be rewritten as:

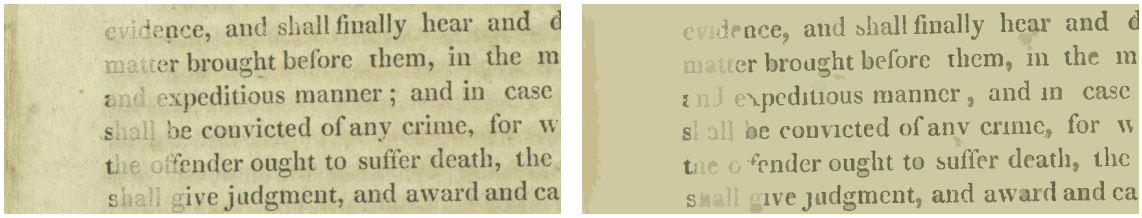
$$\begin{aligned}
\Delta E(X) &= E(\mathcal{S}') - E(\mathcal{S}' \setminus \{X\}) \\
&= \sum_{X \in X_{\text{pr}} \cup Y_{\text{pr}}} ||u(x) - \bar{u}(X_{\text{pr}} \cup Y_{\text{pr}})||_2^2 \; - \sum_{X \in Y_{\text{pr}}} ||u(x) - \bar{u}(X_{\text{pr}})||_2^2 \; - \; \lambda.|\partial X| \\
&= \sum_{x \in X_{\text{pr}}} ||u(x)||_2^2 \; - \; \frac{||\sum_{X_{\text{pr}} \cup Y_{\text{pr}}} u(x)||_2^2}{|X_{\text{pr}}| + |Y_{\text{pr}}|} \; + \; \frac{||\sum_{x \in Y_{\text{pr}}} u(x)||_2^2}{|Y_{\text{pr}}|} \; - \; \lambda.|\partial X|
\end{aligned}$$

From this rewriting, one can see that ΔE can be expressed as a per-shape attribute computation that only requires to sum up $u(x)$, $u(x)^2$ and the number of proper pixels.

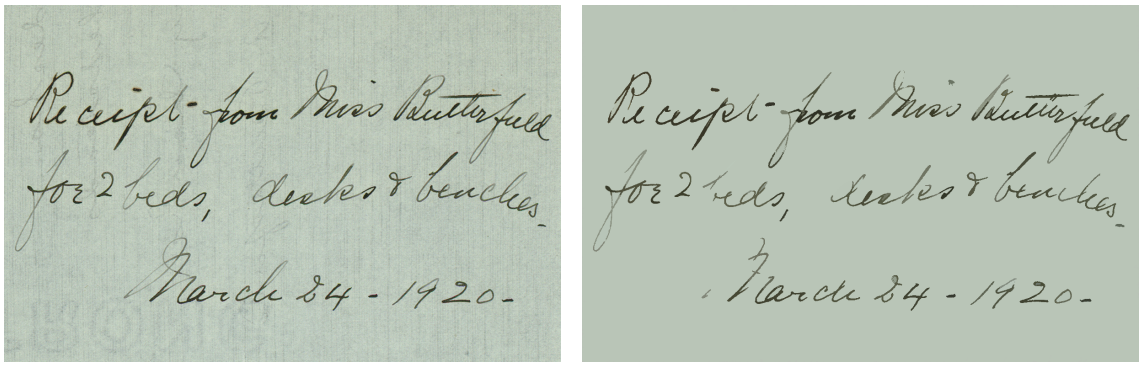
Figure 51: Natural image simplification with the MToS. Left: original images; Right: the simplification running on the MToS. The same λ parameter ($\lambda = 5000$) is used for both images; the simplified images have less than 100 level lines.

10.2.1 *Natural image simplification*

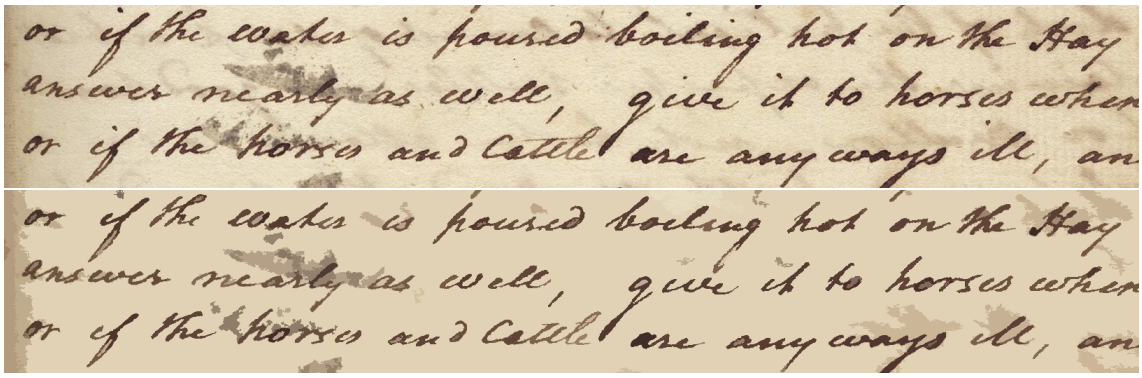
Figure 51 shows the simplification on natural images. While dividing the number of level lines by about 200, the main geometric information is preserved by the simplification. Also, those images are typical cases where the chrominance plays an important role in distinguishing regions with similar brightness. Low-contrasted boundaries between regions of similar luminance create a "leakage" effect and level lines merge unrelated objects that are disjoints with the MToS.



(a) 285 over 113k level lines selected



(b) 112 over 727k level lines selected



(c) 161 over 194k level lines selected

Figure 52: Image simplification for old document restoration.

10.2.2 *Old document restoration*

Figure 52 illustrates the need for contrast invariance in the case of document restoration. Here, the important point is that the MToS is able to retrieve low-contrasted letters even in the presence of "show-through". Since we use a segmentation energy, we do not pretend that it is the perfect solution for document binarization, however since the documents are widely simplified but keep all the objects of interest, it may serve as a pre-processing for a more specific binarization method.

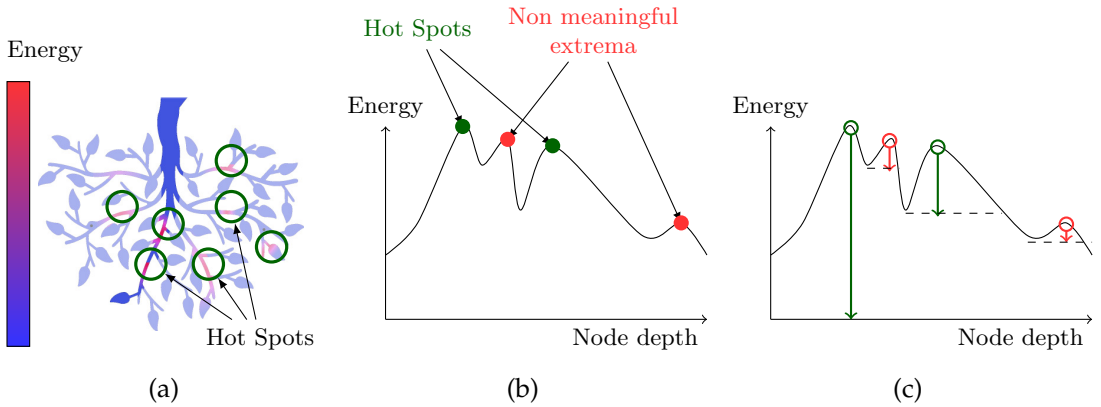


Figure 53: Illustration of getting meaningful high energetic nodes on trees. (a) Schematic view of the energy evolution on the tree nodes. High energetic nodes are located at the same place, in the green circles. (b) Evolution of the energy along a branch of the tree. Interesting nodes are local maxima, however all of them are not interesting. (c) The extinction value (dynamics) of each maximum rendered with arrows, hot spots have a high extinction value compared to non-meaningful maxima.

In chapter 3, we have seen some hierarchical image representations based on level sets: the min-/max-trees and the ToS. An interest of these trees holds in that they enable us to easily filter some components based on some shape descriptors. This is basically a two-step process: first, the features of the shapes are described through a scalar measure (namely, an *attribute*) which is computed on the tree, and second, the nodes below a given threshold are filtered out. In the case of non-increasing attributes (which are the most common), we have seen in section 3.1.2 that there exist several filtering strategies which decide the nodes which will eventually remain.

Let us now consider the case of object detection. While "standard" filtering approaches may yield interesting results, we still face two major problems that are illustrated in fig. 53:

- The shapes evolve slowly along the branch, *i.e.*, there are only few pixels added between a shape and its parent. As a consequence, if we look at the energy distribution on the tree, many close shapes have an high energy and would pass the threshold criterion. In other words, *we would detect the same object several times*. We could limit ourselves to the node with the highest energy per-branch, but then we would face another challenge: what if there are several interesting objects that are nested... They would be in the same branch and we would fail retrieving one of them (see fig. 53a).

- A solution to the previous problem would be to consider only the local maxima of the energy considering the tree topology, *i.e.*, a shape is a local maximum if its energy is greater than the one of its parent and its children. However, even so, many local maxima are located in the same region of the tree. Yet, we would detect the same object several times. In addition, low energetic regions also have non-meaningful local maxima. The fig. 53b shows the energy evolution along a branch. The two first local extrema are located in the same high energetic region, they refer to the same object so one of them is redundant. On the other hand, the last local maximum has a low energy, it is not meaningful.

The approach proposed by Xu et al. [144] is to rely on the dynamics (or more generally on an extinction value) [49, 125] of the extrema to figure out which of them are meaningful. Considering the curve in fig. 53b as a topographical surface, the dynamics is the amount of energy to climb to go from one extremum to another. This is illustrated in fig. 53c. On this example, the non-meaningful extrema have a low extinction value contrary to the ones we want to preserve.

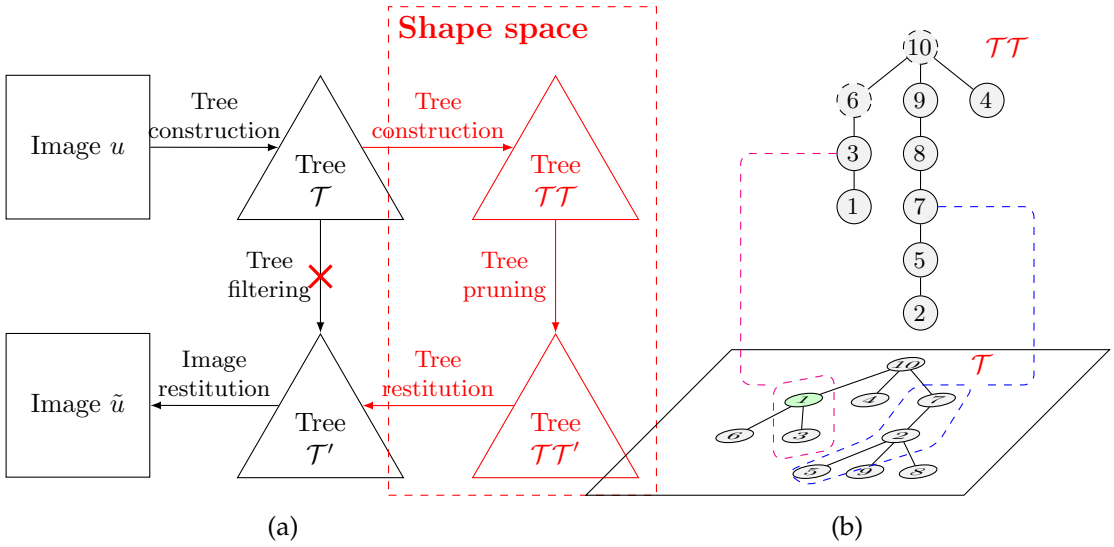


Figure 54: Differences between *shapings* and "classical" filtering. (a) The black path stands for the classical filtering scheme: tree computation, tree filtering, and image restitution. In red is the *shapings*' path. It adds a tree representation of the shape space in which the filtering in performed. (b) Zoom on the *shapings* part. The first tree $\mathcal{T}$ holds components in $\mathcal{P}(\Omega)$. It is valuated with an non-increasing attribute. The second tree $\mathcal{TT}$ is the min-tree of $\mathcal{T}$; its component are sets of shapes, *i.e.*, in $\mathcal{P}(\mathcal{P}(\Omega))$.

An efficient method to perform such a filtering has been introduced in [144] known as *shapings*. This denomination comes from the fact that we do not perform the filtering on the tree $\mathcal{T}$ itself but on second tree $\mathcal{TT}$ that will enable the computation of the extinction values. In other words, we work on the shape space, and the components of the second

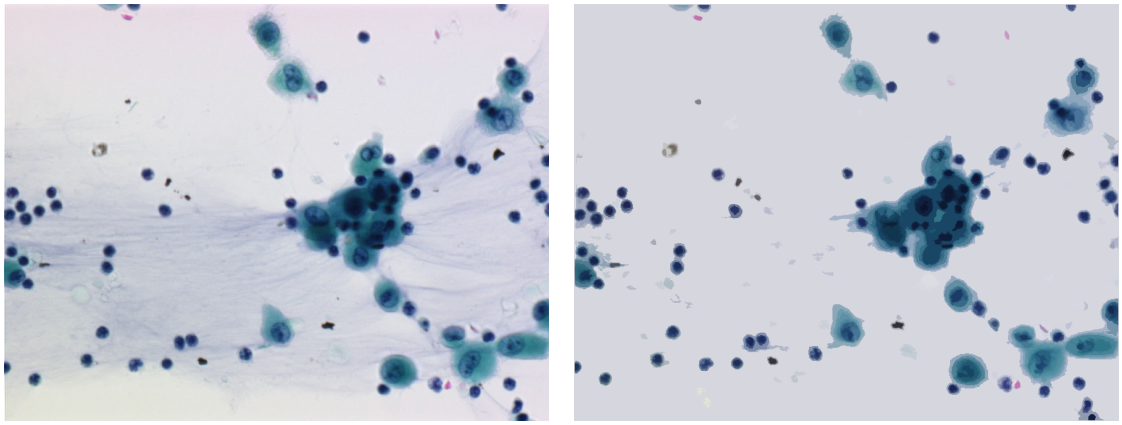


Figure 55: Microscopic image simplification with the MToS using a *shaping* to filter out non-circular objects; from the left image [82], only 650 shapes remain (right).

tree is a set of shapes instead of being a set of pixels. The filtering will thus be performed on $\mathcal{TT}$, it yields $\mathcal{TT}'$. The "proper" elements of the remaining nodes of $\mathcal{TT}'$ give the new set of shapes we want to preserve or remove in $\mathcal{T}$; it gives the new tree $\mathcal{T}'$ from which $\tilde{u}$ can be reconstructed. These differences with a "classical" tree-based attribute filtering is illustrated in fig. 54. Shapings are actually a wider class of *Extinction Filters* that have been shown to be more efficient than standard attribute filters for image registration as they better preserve the structural similarity with the original image [120, 138]. The versatility of the framework allows to use basically any morphological representation for the first structure, e.g. in [144], the authors use the max-tree for blood vessel segmentation, while in [50], the authors use the component graph (see chapter 5) for PET image segmentation. In the following, we share the same idea but use the MToS as the first tree.

10.3.1 *Shapings for Cytology*

In this assessment, we aim at simplifying an image by filtering out objects that do not have a given shape in the context of bronchial cytology. We first valuate a two-term energy $E = E_1 + E_2$ on the tree:

- E_1 expresses the circularity of the shape S. $E_1(S) = 1 - \lambda_2/\lambda_1$ where λ_1 and λ_2 are respectively the lengths of the semi-major axis and semi-minor axes of the best fitting ellipse.
- E_2 expresses the compacity of the shape: $E_2(S) = \text{Perimeter}(S)^2/\text{Area}(S)$.

Then, we look for the shapes that minimize E. As said above, because the energy varies slightly along a branch, we cannot just threshold the energy as it would preserve many close shapes. We thus rely on *shapings* and preserve only the shapes that are local minima. We further simplify the image by computing the extinction values of the minima

and filter out those non-meaningful. Figure 55 shows the simplification on a bronchial microscopical image. As one can see, only nucleus and cytoplasm are well-preserved and the background correctly removed. Such a simplified image can then be combined to a classifier to improve the classification accuracy.

11

IMAGE SEGMENTATION AND CLASSIFICATION

11.1 INTERACTIVE AND AUTOMATIC IMAGE SEGMENTATION

In this section, we show the relevance of the MToS for image segmentation, first in the context of interactive segmentation and then extending the method for unsupervised segmentation where the seeds are detected automatically without requiring any user intervention. First, in section 11.1.1 we present the mainlines of the segmentation method using the tree. Section 11.1.2 shows how our work differs from the other state-of-art object picking methods. Section 11.1.3 gives an in-depth explanation of the proposed method, and in section 11.1.4 we propose a simple extension for non-supervised image segmentation. Eventually, section 11.1.5 shows some results using our proposal and we conclude in section 11.1.6.

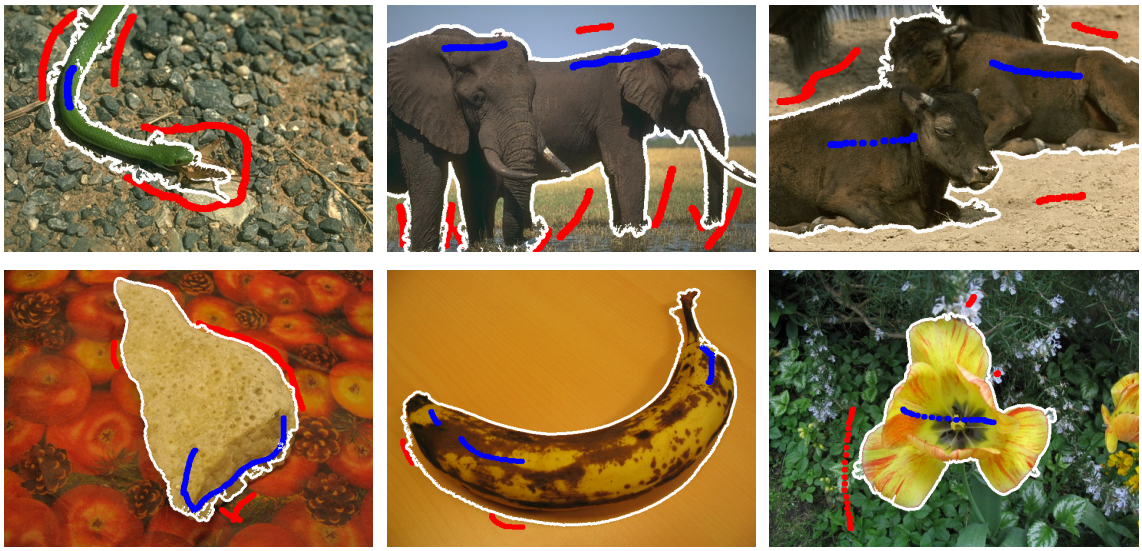


Figure 56: Object picking with our method. Red and blue user scribbles define the background $\mathcal{B}$ and the foreground $\mathcal{F}$ respectively. The white line is the computed $\mathcal{F}/\mathcal{B}$ boundary.

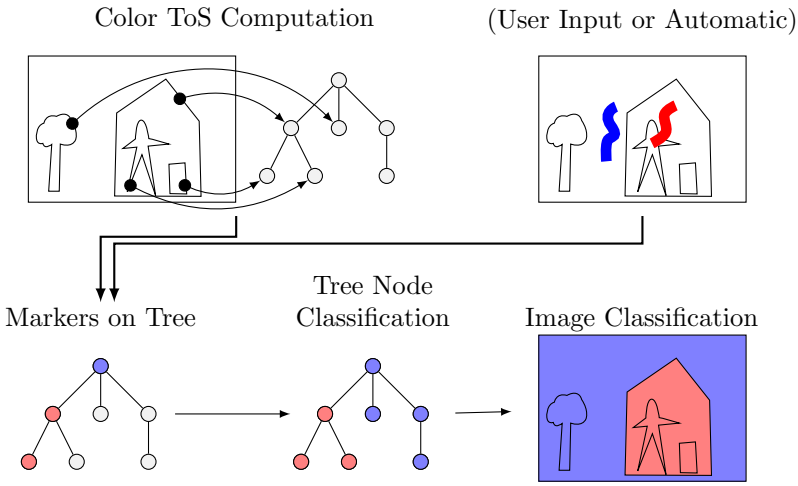


Figure 57: Scheme of the proposed method for object picking.

11.1.1 *Method description*

The problem of object picking can be summarized as follows: given two sets of points F and B in $\mathcal{P}(\Omega)$ representing the user inputs for the *foreground* class ($\mathcal{F}$) and the *background*($\mathcal{B}$), we aim at classifying each point in Ω in one of the two classes.

Consider the distance measure between two points (p, p') in Ω:

$$d_{\mathrm{TV}}(p, p') = \min_{C_{pp'}} \int_0^1 |\nabla u(C_{pp'}(t)).\dot{C}_{pp'}(t)|.dt, \tag{11.1}$$

where $C_{pp'}(t)$ is a path in Ω from p to p'. We can compute the distance between any point x to the seeds F and B and assign the closest class to x. This approach has been used by [107, 12]. However, Dubrovina et al. [44] showed that the proper way to compute such a distance is to have a level set approach and to use the ToS.

Thus, a fundamental idea of the method is to use the MToS representation of the image (instead of working directly on the domain), and to perform the classification on that structure. It applies the same principle as previously (Nearest Neighbor) but uses the tree topology instead of the 2D space topology. The final segmentation is obtained by reconstructing the image from the tree where all nodes have been classified. The method can be summarized in the following steps (see also fig. 57):

1. Compute the MToS $T(\mathbf{u})$ of the image $\mathbf{u}$,
2. Evaluate $T(\mathbf{u})$'s edges with the distance between nodes (see section 11.1.3 for more details),
3. Transpose the user scribbles on $T(\mathbf{u})$, giving two seed sets of node for $\mathcal{F}$ and $\mathcal{B}$,
4. Classify every non-seed node as $\mathcal{F}$ or $\mathcal{B}$ by computing its distance to the seed nodes using $T(\mathbf{u})$'s topology, and retrieving the label of the closest seed node,

Figure 58: Object picking with and without holes. Because we are working in the shape space, tagging the outer object is enough to recover the whole region, but it does not prevent the user from getting objects with holes if he wants to.

5. Reconstruct the image from the labels of $T(\mathbf{u})$,
6. Cleanup: keep significant foreground connected components only.

A significant advantage of working in the shape space is the ability to recover regions of interest that are not even marked by the user. This feature is interesting for objects composed by several other objects. Because a shape is a component without holes, it is enough to select the outer region to retrieve the whole set of objects (see fig. 58).

On the other hand, the approach does not compute any statistics about the regions, but only uses level sets that enable to recover large components with few user scribbles. The amount of markers required actually depends on the number of level lines that separate the background and the foreground.

11.1.2 *Related works*

Our approach is similar to the one proposed by Dubrovina et al. [44] since we both use the ToS (or in our case the MToS) to perform the classification of the nodes and then reconstruct the image from the labeled tree. The main difference is that Dubrovina et al. [44] compute the tree on the likelihood map where each pixel is the confidence to be a foreground pixel. It implies a statistical modeling of the user scribbles and so depends on the accuracy of the modeling of the probability function. Their work is actually an extension of [12] giving a better accuracy for the geodesic distance computation between unlabeled pixels and seeds. In their work, Bai and Sapiro [12] noted how important it is to have an efficient density modeling method that enables a better segmentation while requiring fewer user scribbles than in previous work [107]. But still, the quality of the estimation (so the results) highly depends on the trade-off between the region complexity and the number of user markers. Actually, most image editing or matting state-of-the-art algorithms, including grabcut [110, 67] (GMM modeling) use statistical learning for a background/foreground estimation; our method does not.

11.1.3 *Algorithm details*

The construction of the MToS has been seen chapter 8, we only give a brief summary of it and we mainly detail here the way the ToS is used for segmenting (that are the steps 2–6 of section 11.1.1).

The first step involves the construction of a tree which features similar properties as the ToS but for color images. To that aim, we have introduced in [25] the MToS. Basically, this consists in computing the ToS on each channel independently and merging all shapes in a single structure, called the Graph of Shapes, based on their inclusion. From this graph, a tree is computed through an intermediate depth image that represents the level of inclusion of each pixel in the set of shapes. More details can be found in [25] and in chapter 6.

The second step of the algorithm consists in evaluating a distance on tree edges. As noticed in section 6.3.1, contrary to the standard morphological trees, a node in the MToS may be associated with many colors, as a consequence, we consider the average colors of the points in the node, and the edge between two nodes is evaluated with the distance between their average color (in La*b* space). Then, we need to label the nodes as $\mathcal{F}$ or $\mathcal{B}$ from the user scribbles. In some rare cases, a node may get contradictory labels if the user tags some points as $\mathcal{F}$ and $\mathcal{B}$ that belong to the same node. In that situation, the node should get labeled with the majority class.

The 4^{th} step consists of visiting each non-labeled nodes (shapes) and computing two distances: $d_{\mathcal{F}}$ and $d_{\mathcal{B}}$. These are the distances from S to the nearest foreground and background seed respectively. This can be achieved in two passes. We denote $par(S)$, the parent of S in T, and $d(S, par(S))$ the distance between the average color of S and $par(S)$. At initialization, $d_{\mathcal{B}}(S) = 0$ if the node has background scribbles, $+\infty$ otherwise. The forward step goes from the leaves to the root, and computes:

$$d_{\mathcal{B}}(par(S)) = \min(d_{\mathcal{B}}(par(S)),\, d_{\mathcal{B}}(S) + d(S, par(S)).$$

The backward step goes from the root to the leaves, and computes:

$$d_{\mathcal{B}}(S) = \min(d_{\mathcal{B}}(S),\, d_{\mathcal{B}}(par(S)) + d(S, par(S)).$$

The same computation process holds for $d_{\mathcal{F}}$ and finally, a node gets labeled with $\arg\min_{C\in\{\mathcal{F},\mathcal{B}\}} d_C(S)$.

In the tree $T(\mathbf{u})$, each point belongs to a single node, so the reconstruction of the image consists of assigning to each point the label of the node it belongs to. Eventually, the clean-up step consists of removing each non significant region that is a connected component whose size is below half the size of the largest connected component.

The overall complexity of the method is quasi-linear in the number of points, which is the complexity of computing the MToS [46, 40].

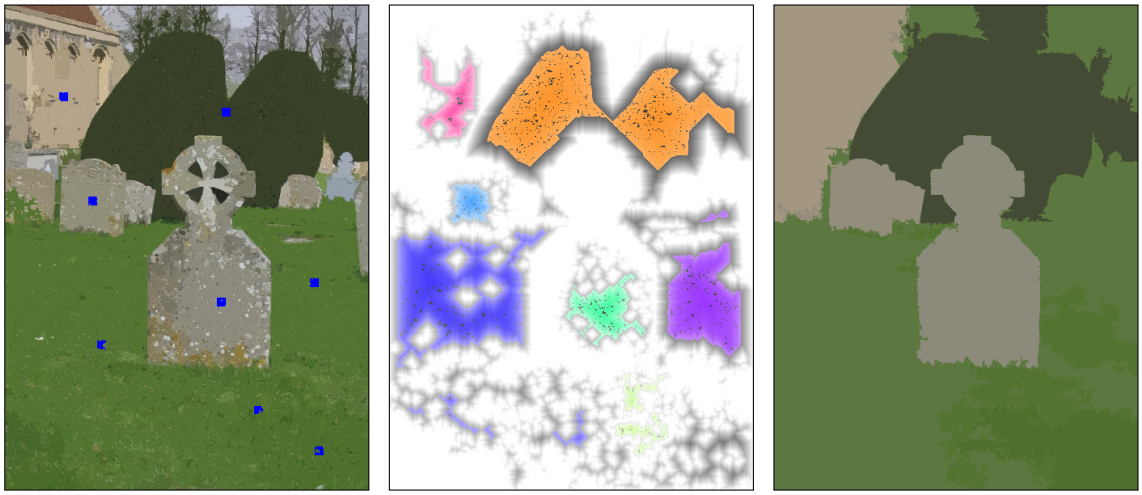


Figure 59: Illustration of the automatic segmentation. Left: Image simplification with the α-tree ($\omega = 200$). Blue squares stand for the centers of the candidate regions ($\lambda = 3000$). Middle: The markers computed over the distance map. Right: The multi-classes segmentation based on the seeds found previously. The segmented components are shown with the average color over the region.

11.1.4 *Extension for automatic segmentation*

The previous framework requires the user to input the scribbles for tagging foreground and background objects. The objective is now to extend this algorithm for automatic segmentation without any user intervention. Instead of a binary segmentation, we first extend the algorithm to a multi-classes classification without any effort. Given a set of marked points $\{M_1, \ldots, M_n\}$ labeled respectively with classes $C_1, \ldots, C_n$, we still aim at classifying any other non-labeled data in Ω. We still use the Nearest Neighbor approach using the distance defined by eq. (11.1). The binary to multi-classes classification is thus straightforward. The real challenge is to automatically get the initial set of seeds. For this purpose, we can rely on any initialization method used by other segmentation algorithms (e.g. the minima of the gradient function as for the Watershed Transform, the modes of the density function as for the Mean Shift algorithm...). We arbitrarily have chosen to initialize our algorithm from the (α, ω)-flat zones of the image that are the set of connected components Γ in the image where the magnitude of the gradient inside Γ cannot exceed α and the amplitude of the range cannot exceed ω [118]. More formally, the initial set of seeds comes from:

$$\{\Gamma_i\} = \{\Gamma \in \bigcup_{\alpha} CC_\alpha(u) \mid \max(u(\Gamma)) - \min(u(\Gamma)) < \omega\}$$

where $CC_\alpha(u)$ denotes the set of α-connected components of $\mathbf{u}$. The set $\{\Gamma_i\}$ forms a partition of the image but leads to an over-segmentation (especially on object boundaries with high gradients), thus we only keep the subset $\{\Gamma'_i\}$ of $\{\Gamma_i\}$ whose size is above a

given threshold. Finally, we compute a thick skeleton of the candidate regions that will be the final markers. The process, illustrated in fig. 59, can be summarized as follows:

1. Compute the α-tree of $\mathbf{u}$ and retrieve the largest α-connected components such that their range does not exceed w and their size is above a given threshold λ.
2. For each component, compute a distance map from the background and threshold this distance map. It yields thick skeletons of the components that are the automatic markers.
3. Perform the multi-classes segmentation extension of the algorithm seen in section 11.1.3.

11.1.5 *Experiments and discussion*

Figures 56 and 60 show some results of the interactive segmentation and automatic segmentation using our approach (implemented using our platform Olena [68, 69]). At this point, and despite its simplicity, our approach for interactive segmentation competes with the state-of-the-art methods such as Grabcut from a qualitative point of view. However it is still premature to have a quantitative assessment with those approaches as we have to tackle some problems first. A common failure case of our method appears when the object of interest is traversed by some level-lines of the background. In that case, we cannot separate correctly the foreground and the background. We have observed in recent experiments that this problem can be partially solved by a different rooting of the MToS (by choosing the point at infinity in the foreground or background scribbles when constructing the tree). The automatic segmentation extension of this work is still in development and is outperformed by the current state-of-art segmentation methods. The method widely depends on the way the seeds are chosen and those given by the α-tree mainly appear in the background. We have to first improve the seed computation process before any further comparison with other methods. However, regarding the quality of the current results, it tends to show the potential of a direct classification in the shape space and that the MToS is an adequate structure for segmentation.

11.1.6 *Conclusion*

We have shown the versatility and the potential of the ToS for segmentation purposes. We have proposed a marker-based classification using the MToS which is free of statistical learning and despite its simplicity, yields results similar to the ones obtained by the state-of-the-art methods. We have also presented an extension that enables an automatic segmentation without requiring any user intervention. The automatic segmentation, still in the process of development, provides interesting results that show the potential of the MToS and the advantages of working in the shape space. As a further work, since our approach is already robust without statistical learning, we plan to introduce

Figure 60: Example of automatic segmentation on the BSD dataset [75]. Left: original images, where blue squares are the markers computed from the α-tree; Right: results of our segmentation method.

more sophisticated classification strategies to improve the interactive segmentation that would enable to further compare our work with other state-of-the-art methods. For the unsupervised segmentation part of this work, since the bottleneck lies in the automatic detection of the seeds, we plan to experiment other pre-processing methods to get better markers.

11.2 DOCUMENT DETECTION IN VIDEOS

In the scope of the ICDAR competition on Smartphone Document Capture and OCR (SmartDoc-2015) [19], we aim at automatically detecting documents in video captured

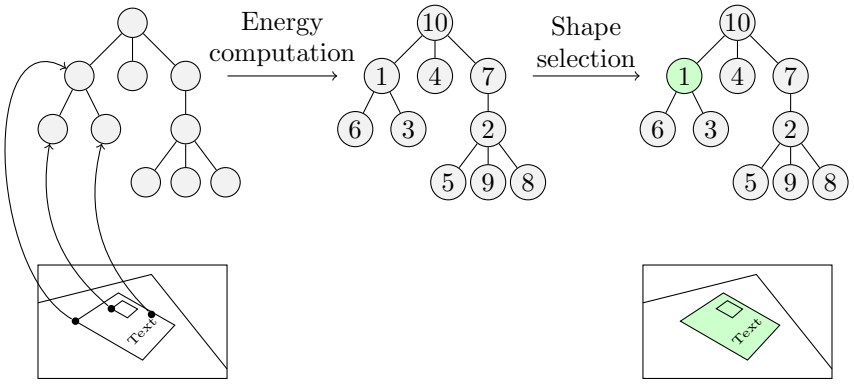


Figure 61: Scheme of the proposed method for document detection in a video frame.

by smartphones. The challenge is motivated by the fact that smartphones are more and more replacing personal scanners as they are affordable and powerful. Nowadays, such acquisition devices are even present in business applications such as document archival, ID scanning, document dematerialization...

The dataset covers different document layout (textual and/or having graphical content) and realistic scene analysis problems (change of illumination, motion blur, change of perspectives, partial occlusions...). There are six different document types that were recorded on four different backgrounds that makes a total of 120 videos and 24.000 frames to process.

In section 11.2.1, we expose our MToS-based method to retrieve a document in a single frame, while in section 11.2.2, we show an extension that enables tracking the document between frames to enhance the detection efficiency. Eventually, in section 11.2.3, we show some results of our method on practical examples and discuss the results.

11.2.1 *Method description*

The method we propose relies on the MToS representation of the image. Basically, we aim at identifying some nodes in the tree that most match some document criteria expressed as shape attributes and as an energy. The two criteria are:

1. How closely the shape boundary fits a quadrilateral, *i.e.*, for each shape A we compute the best fitting quadrilateral $\mathrm{Quad}(A)$ and we measure the ratio:

$$E_1(A) = \frac{|A|}{|\mathrm{Quad}(A)|}$$

2. How "noisy" the object is. We expect a document with some text and graphics, *i.e.*, a shape that contains many shapes inside. At the document level, the letters are

a small grains that look like "noise". Let $\mathcal{L}_A = \{ X \in \mathcal{S} \mid X \subset A \text{ and } X \text{ is a leaf } \}$ denote the set of leaves in the subtree rooted in A, then:

$$E_2(A) = \frac{\sum_{X \in \mathcal{L}_A} (d(X) - d(A))}{|\mathcal{L}_A|}$$

where $d(X)$ stands for the depth of the shape X.

We then look for the shape that minimizes the energy $E(X) = E_1(X) + E_2(X)$. Note that for a better accuracy of E_2, we start with preprocessing the image with a grain filter to ignore the effect of the natural image noise due to the sensor, and so that what we consider as "noise" is big enough to be a letter.

11.2.2 *Tracking document between frames*

When processing the full video clip, one could apply the document detection method exposed in section 11.2.1 on each frame. However, we do not rely on the regularity of the motion that could help to correct miss-detected frames. As we expect small distance between the coordinates of the detection between the current frame and the previous one, the idea is now to retrieve several candidate "document" shapes and using the location of the previously detected document to select the right shapes.

In order to retrieve several candidate shapes from the tree, we cannot just for example select the ten lowest energetic ones. It would lead to the problem exposed in section 10.3. As the energy does not evolve much along a branch, the ten lowest energetic shapes are likely to be located in the same region, thus they refer to the same object. As a consequence, we rely on *shapings* introduced in section 10.3.

First, we retain the 10 best shapes with *shapings*, *i.e.*, the ten local extrema with the highest extinction values, and then, order them w.r.t. their energy E. For the t^{th} frame, we get a family $\{ S_t^1, \dots, S_t^{10} \}$ of candidates shapes. Let S_{t-1}^* denote the shape detected in the previous frame, then the current detected shape S_t^* will be the lowest energetic shape such that its distance with S_{t-1}^* is below a given threshold $d_{\max}$. More formally:

$$S_t^* = S_t^k \,:\, k = \min\{\, i,\, 1 \leq i \leq 10 \mid d(\text{Quad}(S_t^i), \text{Quad}(S_{t-1}^*)) < d_{\max} \,\}$$

where $d(\text{Quad}(X), \text{Quad}(Y))$ is the average euclidean distance between the corners of the best fitting quadrilaterals of X and Y. If no such shape S_t^* exists, the detection fails.

11.2.3 *Results and discussion*

In fig. 62, we show some results of the proposed method in some tricky situations. Indeed, the documents are purposely subject to blur due to camera motion, specular light effects and non-uniform background. These problems are generally fatal to contrast-based methods as they make the document merge with its background. Morphological methods also suffer from these defaults which move the level lines of the image. However even if

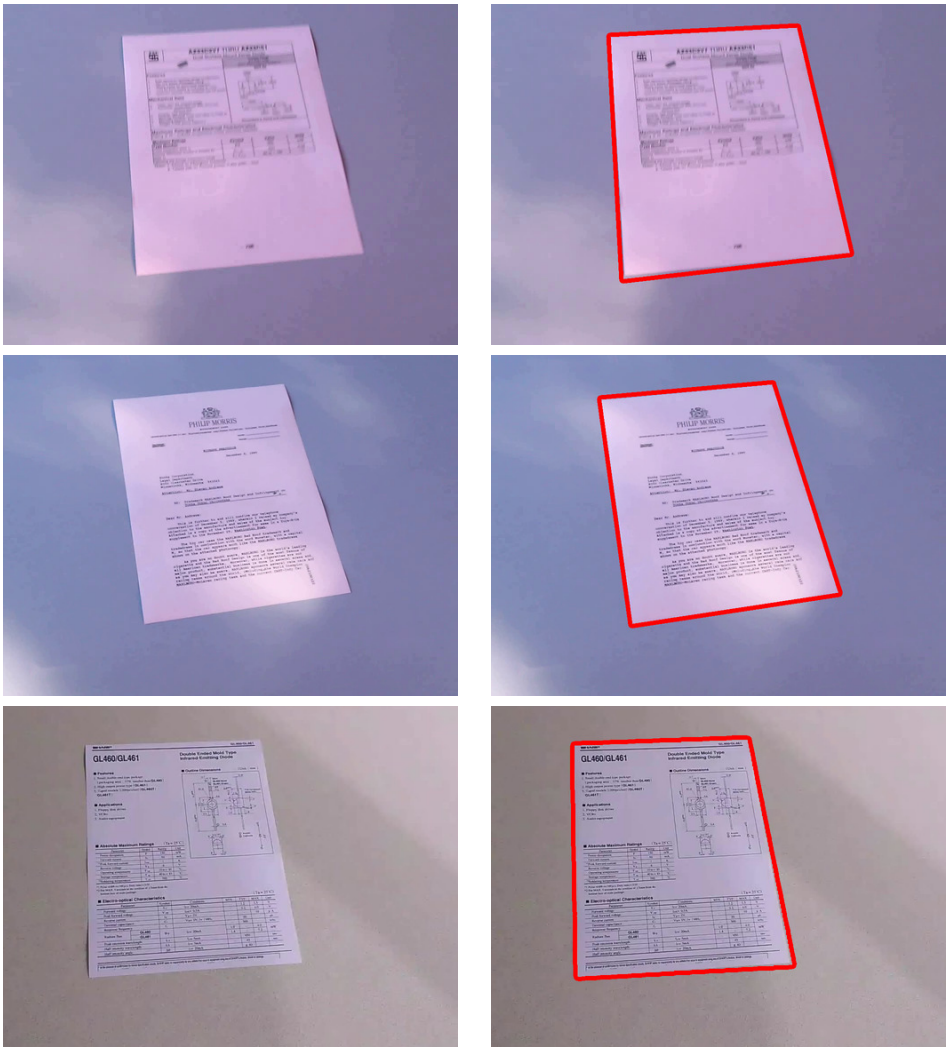


Figure 62: ICDAR competition on document detection. These images show the robustness of our method to blur and light specularity that move object boundaries. Note that some videos are available as supplementary materials [24].

the level line is moved away the real document boundary at some places, some segments are still valid and allow a robust shape analysis, *i.e.*, we are still able to know if the shape looks like a quadrilateral, and then correct *a posteriori* the boundaries with a best quadrilateral fitting.

A common misdetection performed by the method lies in a sub-document retrieval. Indeed, some documents (e.g. forms, tables) may present a layout such that an inner-table may have lower energy than the whole document. For the challenge, this problem has been solved with an *a priori* knowledge about the expected size of the document, but a more robust approach could be to favor the largest shape in the case where two nested shapes have similar energies.

Ranking	Method	Jaccard Index	Confidence Interval
1	LRDE (ours)	0.9716	[0.9710, 0.9721]
2	ISPL-CVML	0.9658	[0.9649, 0.9667]
3	SmartEngines	0.9548	[0.9533, 0.9562]
4	NetEase	0.8820	[0.8790, 0.8850]
5	A2iA run 2	0.8090	[0.8049, 0.8132]
6	A2iA run 1	0.7788	[0.7745, 0.7831]
7	RPPDI-UPE	0.7408	[0.7359, 0.7456]
7	SEECS-NUST	0.7393	[0.7353, 0.7432]

Table 2: Global results for the Smartdoc Challenge 1 competition.

The method (slightly modified with *a priori* knowledge about the documents, e.g. the expected size of the document w.r.t. the resolution of the video) got the first place of the competition among 7 participants (see table 2). The evaluation was based on the Jaccard index, that measures the similarity between the set of expected pixels in the ground truth and the set of the segmentation result returned by the method. Our method obtained an average Jaccard index of 0.9716, varying between 0.9710 and 0.9721 on the whole dataset [19], which tends to show the robustness of the proposed approach.

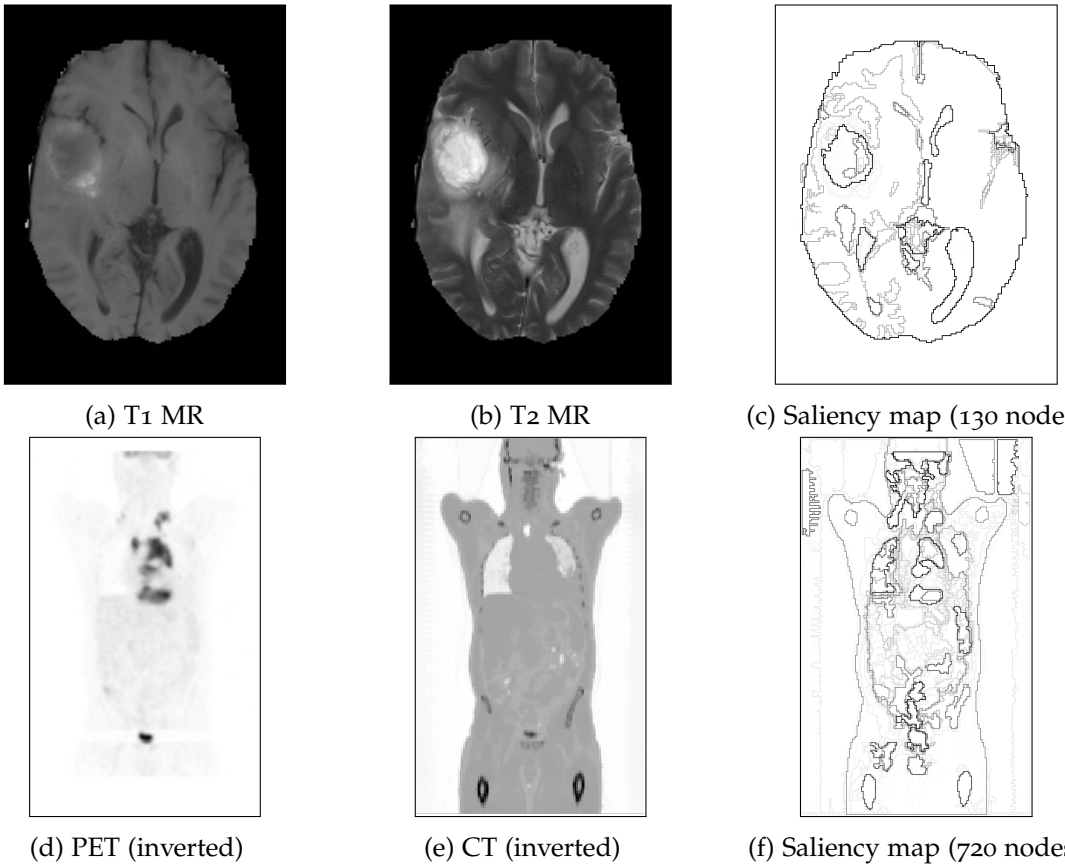
11.3 HIERARCHICAL SEGMENTATION ON MULTIMODAL IMAGES

In [144], in the continuation of their work on *shapings* (see. section 10.3), the authors proposed a generic method to get a hierarchical segmentation of the image from any tree representation.

Contrary to the hierarchies of segmentation seen in chapter 2, any cut in threshold-set-based morphological trees does not lead to a segmentation but rather to a partial partition of the image. Let $\mathcal{A}$ be an increasing attribute over a component tree $\mathcal{T} = (\mathcal{S}, \subseteq)$, without loss of generality, we consider that $\mathcal{T}$ is a tree of shapes. A "cut" in $\mathcal{T}$ at scale λ defines the set of shapes that are minimal elements (w.r.t. the inclusion relation) of $\{ X \in \mathcal{S} \mid \mathcal{A}(X) \geq \lambda \}$. Obviously, a "cut" in those trees yields a set of disjoint connected components $\{ X_1, \ldots, X_n \}$ and one could consider $X_0 = \Omega \setminus \bigcup_{1 \leq i \leq n} X_i$ as the extra background class to get the full partition $\pi_1 = \{ X_0, \ldots, X_n \}$ of the image. However, when considering the partition π_2 of a "cut" of higher scale $\{ Y_1, \ldots, Y_m \}$ and its background class Y_0, then we have:

$$\forall i, 1 \leq i \leq n,\ \exists j, 1 \leq j \leq m,\ X_i \subseteq Y_j$$

but, $X_0 \not\subseteq Y_0$ so π_1 is not a refinement of π_2 and we do not have hierarchy of partition.



(a) T1 MR (b) T2 MR (c) Saliency map (130 nodes)

(d) PET (inverted) (e) CT (inverted) (f) Saliency map (720 nodes)

Figure 63: Saliency map on multimodal medical images.

The idea introduced in [144] consists in the definition of a partition of the image in terms of shape contours. Given a set of shapes $\mathcal{S}' \subseteq \mathcal{S}$, and a Boolean image f defined on the cubical grid $\mathcal{K}_\Omega$ (see. section 7.1) as:

$$f(x) = \begin{cases} 1 & \text{if } \exists X \in \mathcal{S}',\ x \in \partial X \\ 0 & \text{otherwise} \end{cases}$$

then the connected components of $[f < 1] \cap \Omega$ define a partition of Ω. We note $\pi(\mathcal{S})$ the partition associated with the shape set $\mathcal{S}$. Then, π is decreasing as for any $\mathcal{S}_1 \subseteq \mathcal{S}_2$, we have $\pi(\mathcal{S}_2) \sqsubseteq \pi(\mathcal{S}_1)$. The need for *shapings* and working in the shape space is now obvious as we aim at defining a family of decreasing shape set $\{ \mathcal{S}_1, \ldots, \mathcal{S}_k \}$ to build the family $H = \{ \pi(\mathcal{S}_1), \ldots, \pi(\mathcal{S}_k) \}$ which is a hierarchy of partition.

In section 10.3, we have reviewed the basics of *shapings*. Given the attribute (an energy) $\mathcal{A}$ computed over a tree, one can remove the nodes below a given threshold; it yields a simplification of the image. The problem was that there is a redundancy in the remaining shapes since two close nodes are likely to have the same energy, so to pass the criterion. The idea introduced in [144] is to preserve only the local extrema in the tree and compute

their extinction value $\mathcal{AA}$ (the filtering value required so that an extremum gets merged with another extremum). Let the sequence $E = \{ S_1, \ldots, S_n \}$ be the extremal shapes ordered increasingly according to their extinction value. Then, we have $\{ \mathcal{S}_1, \ldots, \mathcal{S}_n \}$ where $\mathcal{S}_k = \{ S_k, \ldots, S_n \}$ which defines a sequence of decreasing shape sets that can be used to build the hierarchy of segmentation. It is commonly represented as a saliency map f where the extinction values computed for each extremum node are set back on the contour of the shapes in the original domain space:

$$f(x) = \max\{\mathcal{AA}(X),\ X \in E \mid x \in \partial X\}$$

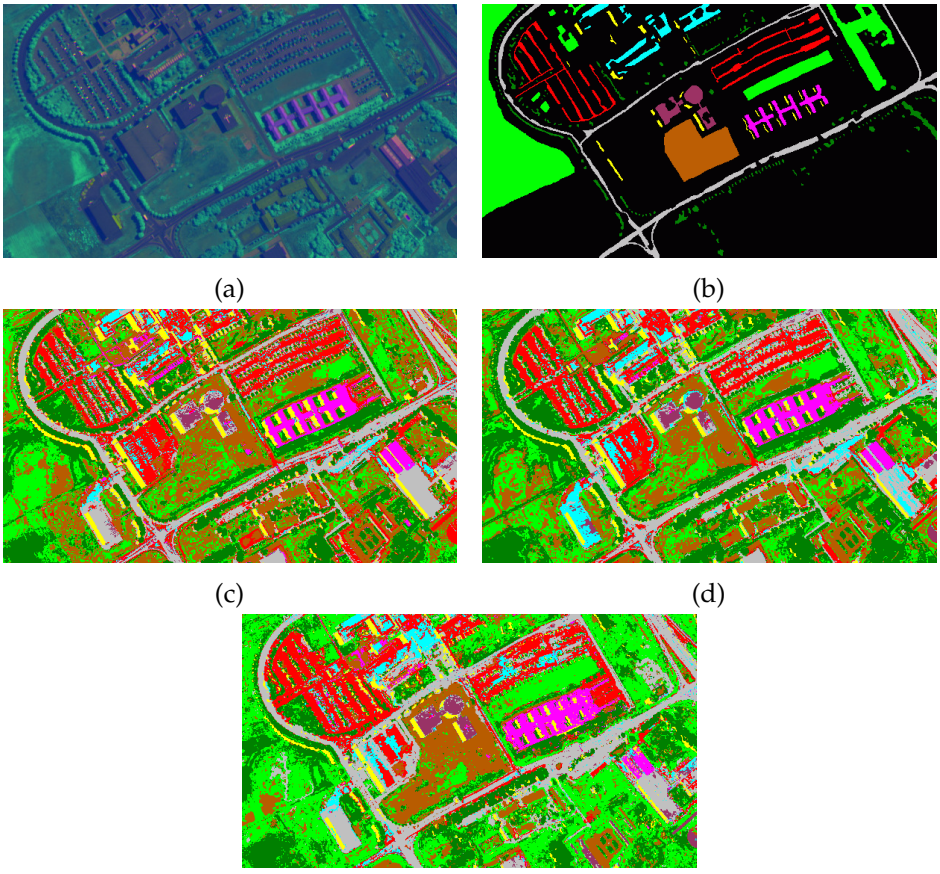
Thresholding the saliency map f with increasing values yields fine to coarse image partitions. Indeed, it is straightforward that the connected components of $[f \leq \lambda]$ actually corresponds to the partition $\pi(\mathcal{S}_i)$ where $i = \max\{ k,\ \mathcal{AA}(S_k) \leq \lambda \}$

Examples of saliency maps are shown in fig. 63, where images have been acquired with different devices (the brain image with T1- and T2- MRI scans and the body image with PET and CT scans). We have computed the MToS on these images and used the MSER criterion as the energy. As a first remark, the MToS preserves the geometric information of the two channels and mixes them in a sensible way. For example, the heart only appears on the PET scan (fig. 63d) and the lungs in the CT scan (fig. 63e) but both appear on saliency map. Second, as one can see the most important objects appear with a high saliency so they will be filtered last on segmentation hierarchy.

11.4 CLASSIFICATION OF HYPERSPECTRAL IMAGES

With the recent advances in optical satellite sensors, the high spatial resolution images issued from these devices have allowed a broader class of applications. A particular interest of the very high resolution imagery has been the classification of urban scenes in order to study their evolution. Practical applications include the automatic update of transport network maps, the discovering of new buildings for fraud detection, the refunding cost estimation after disasters...

In [41], the authors proposed a morphological approach for the classification based on morphological profiles which is a pixel-wise study of the response to a bank of morphological filters [106]. Yet, the processing is marginal. Those filters are applied channel-wise and the responses in each band are concatenated to form a feature space. Here we propose to use filters based on our MToS, *i.e.*, with a vectorial approach. In section 11.4.1, we review the principles of morphological profiles for hyperspectral classification and we expose the integration of the MToS in the processing line. In section 11.4.2, we compare the results obtained by the MToS compared to other state-of-the-art morphological profiles and show the benefits of our structure.



(a) (b) (c) (d) (e)

Figure 64: Hyperspectral image classification. (a) Three principal components recomposed as an RGB image. (b) Ground truth. (c) Classification with AP. (d) Classification with MSDAP. (e) Classification with VSDAP.

11.4.1 *Method description*

Dalla Mura et al. [41] proposed to use morphological attribute profiles to perform the classification of hyperspectral images acquired by Quickbird. The basic idea is to study the behavior at each pixel of an operator at different strengths of filtering. Because no assumption can be made about the type (dark or light) of the objects to detect, they were interested in self-dual profiles. As a consequence, in [41], they compute a set of attribute openings and closings channel-wise at different predefined thresholds that yields the feature space on which they perform the classification. More formally, given an attribute opening γ_λ and the attribute closing ϕ_λ where $\lambda \in \{\lambda_1, \ldots, \lambda_n\}$ is a family of threshold values, they define their feature map ω as:

$$\omega(x) = \{ \gamma_{\lambda_i}(u_k)(x), \phi_{\lambda_i}(u_k)(x) \}_{k,\lambda_i}$$

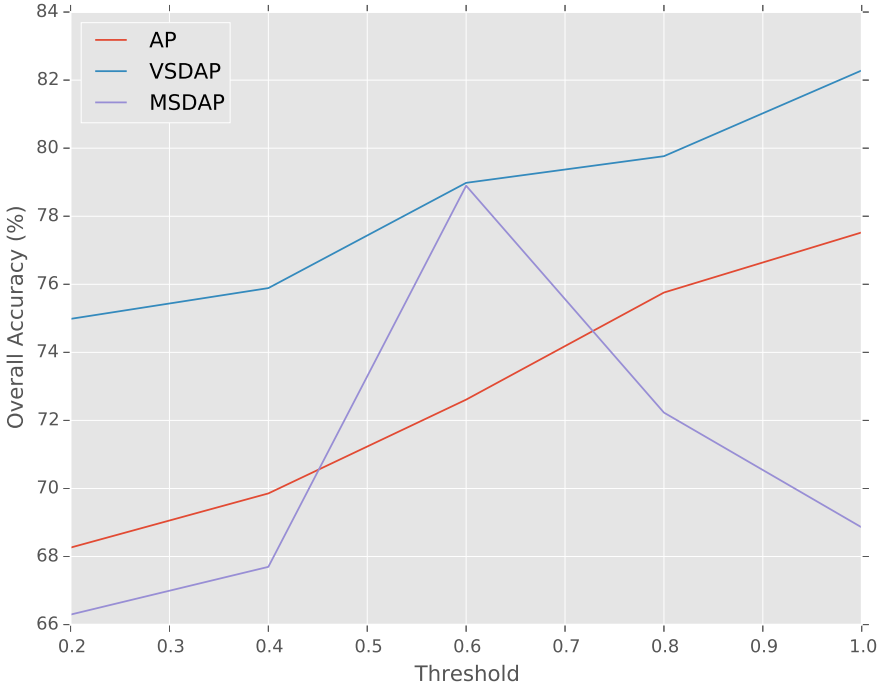


Figure 65: Overall accuracy of the classification by the AP, MSDAP, and VSDAP w.r.t. the number of thresholds used in the profiles (so the dimension of the feature space). The x-axis represents intervals, e.g. the value 0.6 stands for the thresholds $\{0.2, 0.3, \ldots, 0.6\}$.

In [41], they replace the min and max-trees used for the computation of the openings and the closings by self-dual attribute filters using the ToS. They further compare the results in [33] and show that the classification with self-dual attribute profiles outperforms the previous approach with min and max-trees. The features space size is divided by two since each pair of dual filtering $(\gamma_{\lambda_i}(u_k)(x), \phi_{\lambda_i}(u_k)(x))$ is replaced by a single self-dual filtering $\rho_{\lambda_i}(u_k)(x)$.

Yet, the ToS is computed marginally on each channel independently, so we extend their approach with MToS-based filters which yields a non-marginal processing since a single tree representing the image is used. However, the feature space has the same size as for self-dual attribute profile, since the output of each filter ρ is now a vector:

$$\omega(x) = \{\rho_{\lambda_i}(\mathbf{u})(x)_k\}_{k,\lambda_i}$$

11.4.2 *Experiments and discussion*

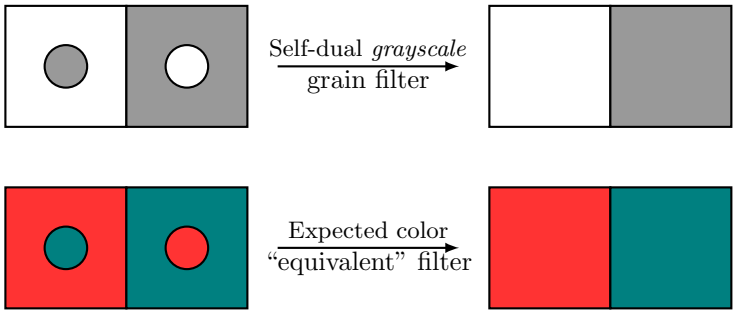
The experiments were carried out on a hyperspectral image acquired by Quickbird of Pavia, Italy. Trees are computed on the first three components of the Principal Component Analysis. The fig. 64 shows the results of the classification with attributes profiles (AP), the marginal self-dual attribute profiles (MSDAP) and the vectorial self-dual attribute profiles (VSDAP). For each method, the same attribute (moment of inertia), the same filtering parameters (0.1, 0.2, ..., 1.0), and the same classifier (Random Forest) are used. Figure 65 shows the overall classification accuracy as a function of the number of thresholds used in the profile. When all thresholds are used, the VSDAP gets an Overall Accuracy of 82.2% while in the same time the AP and MSDAP achieve respectively 77.5% and 68.9%. Using the MToS for filtering and computing the profiles clearly outperforms the ones based on the other trees and tends to confirm that the MToS retrieves and correctly synthesizes the geometric information available in the different channels.

12

CONCLUSION AND PERSPECTIVES

CONCLUSION

To access an high level of understanding of the image, we have motivated the claim that hierarchical representations of image contents are easy-to-use and efficient solutions. To exploit the power of hierarchical structures from the Mathematical Morphology (MM) framework, image values need to be ordered. Even the most basic operations from the MM (*e.g.* dilation and erosion) require a special organization of values (namely, a complete lattice). In other words, the values themselves do not mean anything, only the way they are related matters. However, when dealing with multivariate images, a total ordering of data is usually not satisfactory. In addition, when considering self-dual filters, we can intuitively expect the output of such a filter without ordering data as shown below. Here, the notion of small objects (grain) is *a priori* not dependent on the ordering. So fortunately, we avoid to compare/order red and blue.



As a consequence, there is no reason to rely on an order that would mix non-comparable components if it does not make sense. On the contrary, if an ordering of values inside each channel makes sense, we should rely on it. In addition, channel-wise, we would like to preserve important properties of morphological self-dual operators, *i.e.*, the invariance to any marginal change of contrast and any marginal inversion of contrast (self-duality in the multidimensional case).

The Multivariate Tree of Shapes (MToS) presented here is a novel representation that extends the grayscale ToS on multivariate images. This representation relies on the ToS's computed marginally on each channel of the image. It merges the marginal shapes in a "sensible" way by preserving the maximum number of inclusion. The method proposed has theoretical foundations expressing the ToS in terms of topographic map of the total variation computed from the image border. This reformulation has allowed its extension on multivariate data. We also have proven that the tree provided by our method meets correctly those properties. To close the theoretical part, we have carried out experiments

testing the robustness of the structure to non-relevant components (*e.g.* with noise or with low dynamics) and we showed that such defaults do not affect the overall structure of the MToS.

Because this work took place in a context where the practical usage is of prime importance, we have also been involved in the validation of the soundness of our approach on many real-case applications. Some of them were just an adaptation of ToS-based methods to the new structure. We succeed in using the MToS for image filtering, image simplification, image segmentation, image classification and object detection. The use of the MToS generally outperforms its ToS-based counterpart, showing the potential of our approach.

Eventually, because new real-case applications require processing more and more data in a short time, we have been involved in developing a fast algorithm to compute the MToS, as well as some tree processing algorithms (see part iv). We have studied their complexity and have shown that the MToS can be built in quasi-linear time w.r.t. the number of pixels and is quadratic w.r.t. the number of channels.

EXTENSION AND FUTURE WORK

Multivariate Component-trees

The method proposed in chapter 6 is a two-step process where the first part consists in merging marginal ToS's into a single structure, the GoS, and the second part consists of extracting a tree from the graph. This framework can be generalized for other trees as inputs. In particular, nothing prevents the user from providing min- or max-trees as inputs, before computing the graph of inclusion. The method would then provide a tree which is marginally invariant to change of contrast. It may serves to define Multivariate Min- or Max-trees (depending on the trees' type in input) and so, a new approach to extend component-trees to multivariate images *without requiring the imposition of a total ordering*. Consequently, it may also be the base to define new connected operators, *e.g.* openings or closings, on multivariate images.

Algebraic properties of a MToS-based filter

For many applications (*e.g.* object detection, segmentation) the image reconstruction process is not necessary as we just need to identify or label some nodes of the tree. However, in the case of image filtering (*e.g.* denoising), the reconstruction is a fundamental step. With the MToS, a node may be associated with different values (just like component-trees in the case of total preorders). It leads to a problem during the reconstruction after filtering as one as to decide of a filtering value (see. sections 4.3.6 and 6.3.1). A straightforward solution that we have adopted is to assign the average vector to each node. However, one can wonder about the algebraic properties of such reconstruction. In particular, given the MToS $T(\mathbf{u})$ of $\mathbf{u}$ followed by the reconstruction of $\tilde{\mathbf{u}}$, it is not

evident that $T(\mathbf{u}) = T(\tilde{\mathbf{u}})$. This is a problem for defining "real" morphological filters like a Multivariate grain filters. Indeed, we cannot ensure that the MToS of the reconstructed image from a filtered tree are the same. In other words, two consecutive filtering with the same grain size may give different results; we have lost the idempotency. Few authors have been studying this problem, but it also arises with component-trees based on image-dependent total orderings since the ordering built from the first image is not the same as the one built on the reconstructed image.

Part IV

APPENDICES

A

A COMPARATIVE REVIEW OF COMPONENT TREE COMPUTATION ALGORITHMS

Edwin Carlinet and Thierry Géraud. "A Comparative Review of Component Tree Computation Algorithms". In: *IEEE Transactions on Image Processing* 23.9 (Sept. 2014), pp. 3885–3895.

A.1 INTRODUCTION

In mathematical morphology, connected filters are those that modify an original image by only merging flat zones, hence those that preserve some of the original image contours. Originally, they were mostly used for image filtering [130, 112]. Major advances came from max- and min-tree as hierarchical representations of connected components and from an efficient algorithm able to compute them [114]. Since then, usage of these trees has soared for more advanced forms of filtering: based on attributes [60, 55], using new filtering strategies [114, 124], allowing new types of connectivity [99]. They are also a base for other image representations. In [85] a tree of shapes is computed from a merge of the min- and max- trees. In [141] a component tree is computed over the attributes values of the max-tree. Max-trees have been involved in many applications: computer vision through motion extraction [114], features extraction with MSER [76], segmentation, 3D visualization [132]. With the increase of applications comes an increase of data type to process: 12-bit images in medical imagery [132], 16-bit or float images in astronomical imagery [15], and even multivariate data with special ordering relation [104]. With the improvement of optical sensors, images are getting bigger (so do image data sets) which argues for the need for fast algorithms. Many algorithms have been proposed to compute the max-tree efficiently but only partial comparisons have been proposed. Moreover, some of them are dedicated to a particular task (e.g., filtering) and are unusable for other purposes.

In a short paper [27], we have presented a first comparison of many state-of-the-art max-tree algorithms in a unique framework, i.e., same architecture, same language (C++) and same outputs. Yet this comparison was performed on a single image, the pseudo-code of all the algorithms were not listed, and the description of those algorithms and their comparison were short. This paper aims at correcting those three drawbacks, so it presents a full and exhaustive comparative review of the state-of-art component tree computation algorithms.

The paper is organized as follows. Appendix A.2 recalls basic notions and manipulations of max-tree. Appendix A.3 describes the algorithms and implementations used in this study; in particular, a new technique that improves the efficiency of union-find-based algorithms is presented in Appendix A.3.1.4. Appendix A.4 is dedicated to the comparison of those algorithms both in terms of complexity and running times through experimentations. Last we conclude in Appendix A.5.

A.2 A TOUR OF MAX-TREE: DEFINITION, REPRESENTATION AND ALGORITHMS

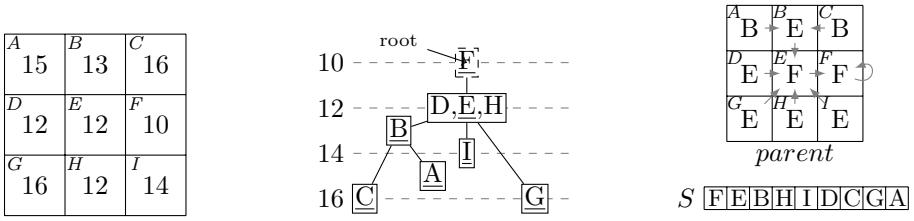
A.2.1 *Basic notions for max-tree*

Let $f : \Omega \to V$ be an image on a regular domain Ω, having values on a totally preordered set $(V, \leq)$ and let $\mathcal{N}$ be a neighborhood on Ω. Let $\lambda \in V$, we note $[f \leq \lambda]$ the set $\{ p \in \Omega,\ f(p) \leq \lambda \}$. Let $X \subset \Omega$, we note $CC(X) \subset \mathcal{P}(\Omega)$ the set of connected components of X w.r.t. the neighborhood $\mathcal{N}$; $\mathcal{P}(\Omega)$ being the power set of all the possible subsets of Ω. $\{ CC([f = \lambda]),\ \lambda \in V \}$ are *level components* and $\Psi = \{ CC([f \geq \lambda]),\ \lambda \in V \}$ (resp. $\leq$) is the set of upper components (resp. lower components). The latter endowed with the inclusion relation form a tree called the max-tree (resp. min-tree). Since min- and max-trees are dual, this study obviously holds for min-tree as well. Finally, the peak component of p at level λ noted P_p^λ is the upper component $X \in CC([f \geq \lambda])$ such that $p \in X$.

A.2.2 *Max-tree representation*

Berger et al. [15], and Najman and Couprie [91] rely on a simple and effective encoding of component-trees using an image that stores the *parent* relationship. The latter exists between two components A and B whenever A is *directly* included in B (*parent* is actually the covering relation of $(\Psi, \subseteq)$). An upper component is represented by a single point called the *canonical element* [15, 91] or *level root*. Let two points $p, q \in \Omega$, and p_r be the root of the tree. We say that p is canonical if $p = p_r$ or $f(parent(p)) < f(p)$. A *parent* image shall satisfy the following three properties: 1) $parent(p) = p \Rightarrow p = p_r$ - the root points to itself and it is the only point verifying this property - 2) $f(parent(p)) \leq f(p)$ and 3) $parent(p)$ is canonical.

Furthermore, having just the *parent* image is an incomplete representation since it is not sufficient to easily perform classical tree traversals. For that, we need an extra array of points, $S : \mathbb{N} \to \Omega$, where points are stored so that $\forall i, j \in \mathbb{N}\ i < j \Rightarrow S[j] \neq parent(S[i])$. Thus browsing S elements allows to traverse the tree downwards i.e from the top (the root) to the bottom of the tree (the leaves). On the contrary, a reverse browsing of S is an upward tree traversal. Note that having both S and *parent* thus makes it useless to store the children of each node. Figure 66 shows an example of such a representation of a max-tree. This representation only requires $2nI$ bytes memory space where n is the number of pixels and I the size in bytes of an integer, since points stored in S and *parent*



(a) Original image f (b) Max-tree of f (c) Max-tree representation

Figure 66: Representation of a max-tree using the 4-connectivity with a parent image and an array. Canonical elements are underlined.

are actually positive offsets in a pixel buffer. The algorithms we compare have all been modified to output the same tree encoding, that is, the couple (*parent*, *S*).

A.2.3 *Attribute filtering and reconstruction*

A classical approach for object detection and filtering is to compute some features called attributes on max-tree nodes. A usual attribute is the number of pixels in components. Followed by a filtering, it leads to the well-known area opening. More advanced attributes have been used like elongation, moment of inertia [134] or even Mumford-Shah like energy [141]. Some max-tree algorithms [135, 76] construct the *parent* image only; they do not compute *S*. As a consequence, they do not provide a "versatile" tree, i.e., a tree that can be easily traversed upwards and downwards, that allows attribute computation and non-trivial filtering. Here we require every algorithms to output a "complete" tree representation (*parent and S*) so that it can be multi-purposedly usable. The rationale behind this requirement is that, for some applications, filtering parameters are not known yet at the time the tree is built (e.g., for interactive visualization [132]). In the algorithms we compare in this paper, no attribute computation nor filtering are performed during tree construction for clarity reasons; yet they can be augmented to compute attribute and filtering at the same time. Algorithm 5 provides an implementation of attribute computation and direct-filtering with the representation. $\hat{a} : \Omega \times V \to \mathcal{A}$ is an application that projects a pixel p and its value $f(p)$ in the attribute space $\mathcal{A}$. $\hat{+} : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ is an associative operator used to merge attributes of different nodes. `compute-attribute()` starts with computing attributes of each singleton node and merges them from leaves toward root. Note that this simple code relies on the fact that a node receives all information from its children before passing its attribute to the parent. Without any ordering on *S*, it would not have been possible. `direct-filter()` is an implementation of direct filtering as explained in [114] that keeps all nodes passing a criterion λ and lowers nodes that fail to the last ancestor "alive". This implementation has to be compared with the one in [135] that only uses *parent*. This one is shorter, faster and clearer above all.

Algorithm 5: Attribute computation and filtering algorithms.

function `compute-attribute`$(S, parent, f)$
- $p_{root} \leftarrow S[0]$;
- **foreach** $p \in S$ **do** $attr(p) \leftarrow \hat{a}(p, f(p))$;
- **foreach** $p \in S$ *backward*, $p \neq p_{root}$ **do**
  - $q \leftarrow parent(p)$;
  - $attr(q) \leftarrow attr(q) \hat{+} attr(p)$;
- **return** $attr$;

function `direct-filter`$(S, parent, f, attr)$
- $p_{root} \leftarrow S[0]$;
- **if** $attr(p_{root}) < \lambda$ **then** $out(p_{root}) \leftarrow 0$;
- **else** $out(p_{root}) \leftarrow f(p_{root})$;
- **foreach** $p \in S$ *forward* **do**
  - $q \leftarrow parent(p)$;
  - **if** $f(q) = f(p)$ **then** $out(p) \leftarrow out(q)$; `/* (1) */`
  - **else if** $attr(p) < \lambda$ **then** $out(p) \leftarrow out(q)$; `/* (2) */`
  - **else** $out(p) \leftarrow f(p)$; `/* (3) */`
- **return** out;

(1) p not canonical, (2) Criterion failed, (3) Criterion passed

A.3 MAX-TREE ALGORITHMS

Max-tree algorithms can be classified in three classes:
Immersion algorithms. They start with building N disjoint singletons, one for each pixel and sort them according to their gray value. Then, disjoint sets merge to form a tree using the union-find algorithm [4, 122].
Flooding algorithms. A first scan allows to retrieve the root which is a pixel at lowest level in the image. Then, they perform a propagation by flooding first the neighbor at highest level i.e. a depth first propagation [114, 133].
Merge-based algorithms. They divide an image in blocks and compute the max-tree on each sub-image using another max-tree algorithm. Sub max-trees are then merged to form the tree of the whole image. Those algorithms are well-suited for parallelism using a *map-reduce* (or *divide-and-conquer*) approach [100, 135]. When blocks are image lines, dedicated 1D max-tree algorithms can be used [77, 81, 86].

A.3.1 *Immersion algorithms*

A.3.1.1 *Basic principle*

Berger et al. [15], Najman and Couprie [91] proposed two algorithms based on Tarjan's union-find. They consist in tracking disjoints connected components and merge them in a bottom-up fashion. First, pixels are sorted in an array S where each pixel p represent

the singleton set $\{p\}$. Then, they process pixels of S in backward order. When a pixel p is processed, it looks for already processed neighbors ($\mathcal{N}(p)$) and merges with neighboring connected components to form a new connected set rooted in p. The merging process consists in updating the *parent* pointer of neighboring component roots toward p. Thus, the union-find relies on three processes: `make-set(parent, x)` that builds the singleton set $\{x\}$, `find-root(parent, x)` that finds the root of the component that contains x, and `merge-set(parent, x, y)` that merges components rooted in x and y and set x as the new root. Based on the above functions, a simple max-tree algorithm is given below:

Algorithm 6: Scheme of a union-find-based max-tree algorithm.

```
function Maxtree(f)
    S ← sort pixels increasing;
    foreach p ∈ S backward do
        make-set(parent, p);
        foreach n ∈ N_p processed do
            r ← find-root(parent, n);
            if r ≠ p then merge-set(parent, p, r);
```

`find-root` is a $O(n)$ function that makes the above procedure a $O(n^2)$ algorithm. Tarjan [122] discussed two important optimizations to avoid a quadratic complexity: root path compression and union-by-rank.

When *parent* is traversed to get the root of the component, points of the path used to find the root collapse to the root of the component. However, path compression should not be applied on *parent* image because it removes the hierarchical structure of the tree. As consequence, path compression is applied on an intermediate image *zpar* that stores the root of disjoints components. Path compression bounds union-find complexity to $O(n \log n)$ and has been applied in [15] and [91].

When merging two components A and B, we have to select one of the roots to represent the newly created component. If A has a *rank* greater than B then $root_A$ is selected as the new root, $root_B$ otherwise. When rank matches the depth of trees, it enables tree balancing and guaranties a $O(n \log n)$ complexity for union-find. When used with path compression, it allows to compute the max-tree in quasi-linear time ($O(n.\alpha(n))$ where $\alpha(n)$ is the inverse of Ackermann function which is very low-growing). Union-by-rank has been applied in [91].

Note that *parent* and *zpar* encode two different things, *parent* encodes the max tree while *zpar* tracks disjoints set of points and also uses a tree. Thus, union-by-rank and root path compression shall be applied on *zpar* but never on *parent*.

The algorithm given in Algorithm 7 is the union-find-based max-tree algorithm as proposed by [15]. It starts with sorting pixels that can be done with a counting sort algorithm for low-quantized data or with a radix sort-based algorithm for high quantized data [5].Then it annotates all pixels as *unprocessed* with -1 (in common implementations, pixels are positive offsets in a pixel buffer). Later in the algorithm, when a pixel p is

Algorithm 7: Union-find without union-by-rank.

```
function Find-root(par, p)
    if par(p) ≠ p then par(p) ← find-root(par, par(p));
    return par(p)
function Maxtree(f)
    foreach p do parent(p) ← −1;
    S ← sort pixels increasing;
    foreach p ∈ S backward do
        parent(p) ← p; zpar(p) ← p;                          /* make-set */
        foreach n ∈ N_p such that parent(n) ≠ −1 do
            r ← find-root(zpar, n);
            if r ≠ p then zpar(r) ← p; parent(r) ← p          /* merge-set */ ;
    Canonicalize(parent, S);
    return (parent, S);
function Canonicalize(f, parent, S)
    foreach p in S forward do
        q ← parent(p);
        if f(q) = f(parent(q)) then parent(p) ← parent(q);
```

processed it becomes the root of the component i.e $parent(p) = p$ with $p \neq -1$, thus testing $parent(p) \neq -1$ stands for *is p already processed*. Since S is processed in reverse order and `merge-set` sets the root of the tree to the current pixel p ($parent(r) \leftarrow p$), it ensures that the parent p will be seen before its child r when traversing S in the direct order.

A.3.1.2 *Union-by-rank*

The algorithm given in Algorithm 8 is similar to the one in Algorithm 7 but augmented with union-by-rank. It first introduces a new image *rank*. The `make-set` step creates a tree with a single node, thus with a rank set to 0. The *rank* image is then used when merging two connected sets in $zpar$. Let z_p be the root of the connected component of p, and z_n be the root of connected component of $n \in \mathcal{N}(p)$. When merging two components, we have to decide which of z_p or z_n becomes the new root w.r.t their rank. If $rank(z_p) < rank(z_n)$, z_p becomes the root, z_n otherwise. If both z_p and z_n have the same rank then we can choose either z_p or z_n as the new root, but the rank should be incremented by one. On the other hand, the relation *parent* is unaffected by the union-by-rank, p becomes the new root whatever the rank of z_p and z_n. Whereas without balancing the root of any point p in $zpar$ matches the root of p in parent, this is not the case anymore. For every connected components we have to keep a connection between the root of the component in $zpar$ and the root of max-tree in *parent*. Thus, we introduce an new image *repr* that keeps this connection updated.

Algorithm 8: Union-find with union-by-rank

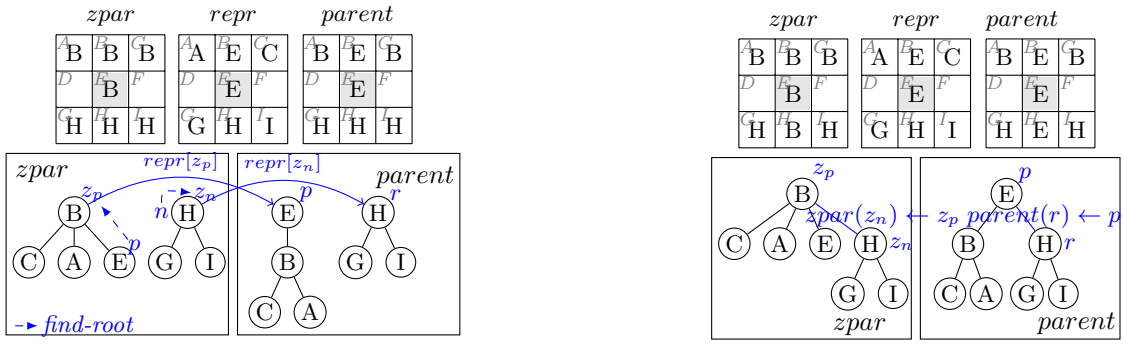
```
function Maxtree(f)
    foreach p do parent(p) ← −1;
    S ← sort pixels increasing;
    foreach p ∈ S backward do
        parent(p) ← p; zpar(p) ← p ;                                  /* make-set */
        rank(p) ← 0; repr(p) ← p; z_p ← p;
        foreach n ∈ N_p s.t. parent(n) ≠ −1 do
            z_n ← find-root(zpar, n);
            if z_n ≠ z_p then
                parent(repr(z_n)) ← p;
                if rank(z_p) < rank(z_n) then swap(z_p, z_n);
                zpar(z_n) ← z_p; repr(z_p) ← p ;                      /* merge-set */
                if rank(z_p) = rank(z_n) then rank(z_p) ← rank(z_p) + 1;
    Canonicalize(parent, S);
    return (parent, S)
```

The union-by-rank technique and structure update are illustrated in Figure 67. The algorithm has been running until processing E at level 12, the first neighbor B has already been treated and neighbors D and F are skipped because not yet processed. Thus, the algorithm is going to process the last neighbor H. z_p is the root of p in $zpar$ and we retrieve the root z_n of n with `find-root` procedure. Using *repr* mapping, we look up the corresponding point r of z_n in *parent*. The tree rooted in r is then merged to the tree rooted in p (*parent* merge). Back in $zpar$, the components rooted in z_p and z_n merge. Since they have the same rank, we choose arbitrary z_p to be the new root.

The algorithm in Algorithm 8 is slightly different from the one of [91]. They use two union-find structures, one to build the tree, the other to handle flat zones. In their paper, `lowernode[`z_p`]` is an array that maps the root of a component z_p in *zpar* to a point of current level component in *parent* (just like `repr(`z_p`)` in our algorithm). Thus, they apply a second union-find to retrieve the canonical element. This extra union-find can be avoided because `lowernode[x]` is already a canonical element, thus $findoot$ on $lowernode(z_p)$ is useless and so does *parent* balancing on flat zones.

A.3.1.3 *Canonicalization*

Both algorithms call the `Canonicalize()` procedure to ensure that any node's parent is a canonical node. In Algorithm 7, canonical property is propagated downward. S is traversed in direct order such that when processing a pixel p, its parent q has the canonical property that is $parent(q)$ is a canonical element. Hence, if q and $parent(q)$ belongs to the same node i.e $f(q) = f(parent(q))$, the parent of p is set to the component's canonical element: $parent(q)$.



(a) (b)

Figure 67: Union-by-rank. (a) State of the algorithm before processing the neighbor H from E. (b) State of the algorithm after processing.

A.3.1.4 *Level compression*

Union-by-rank provides time complexity guaranties at the price of an extra memory requirement. When dealing with huge images, it results in a significant drawback (e.g. RAM overflow...). Since the last point processed always becomes the root, union-find without rank technique tends to create a degenerate tree in flat zones. Level compression avoids this behavior by a special handling of flat zones. In Algorithm 9, p is the point in process at level $\lambda = f(p)$, n a neighbor of p already processed, z_p the root of P_p^λ (at first $z_p = p$), z_n the root of P_n^λ. We suppose $f(z_p) = f(z_n)$, thus z_p and z_n belong to the same node and we can choose any of them as a canonical element. Normally p should become the root with child z_n but level compression inverts the relation, z_n is kept as the root and z_p becomes a child. Since *parent* may be inverted, S array is not valid anymore. Hence S is reconstructed, as soon as a point p gets attached to a root node, p will not be processed anymore so it is inserted in back of S. At the end S only misses the tree root which is $parent[S[0]]$.

A.3.2 *Flooding algorithms*

A second class of algorithms, based on flooding, contrasts with the immersion-based algorithms described in the previous section A.3.1. Salembier et al. [114] proposed the first efficient algorithm to compute the max-tree. A propagation starts from the root that is the pixel at lowest level l_{min}. Pixels in the propagation front are stored in a hierarchical queue composed by as many FIFO queues as the number of levels. It allows to access directly any pixel in the FIFO queue at a given level. Algorithm 10 shows a slightly modified version of Salembier's original algorithm where the original *STATUS* image is replaced by the *parent* image having the same role. The `flood(`λ`, r)` procedure is in charge of flooding the peak component P_r^λ and building the corresponding sub max-tree

Algorithm 9: Union-find with level compression.

```
function Maxtree(f)
    foreach p do parent(p) ← −1;
    S ← sort pixels increasing;
    j = N − 1;
    foreach p ∈ S backward do
        parent(p) ← p; zpar(p) ← p; z_p ← p ;                    /* make-set */
        foreach n ∈ N_p s.t. parent(n) ≠ −1 do
            z_n ← find-root(zpar, n);
            if z_p ≠ z_n then
                if f(z_p) = f(z_n) then swap(z_p, z_n);
                zpar(z_n) ← z_p; parent(z_n) ← z_p ;              /* merge-set */
                S[j] ← z_n; j ← j − 1;
    S[0] ← parent[S[0]];
    Canonicalize(parent, S);
    return (parent, S)
```

rooted in r. It proceeds as follows: first pixels at level λ are retrieved from the queue, their *parent* pointer is set to the canonical element r and their neighbors n are analyzed. If n is not in the queue and has not yet been processed, then n is pushed in the queue for further processing and n is marked as processed ($parent(n)$ is set to `INQUEUE` which is any value different from -1). If the level l of n is higher than λ then n is in the childhood of the current node, thus flooding is halted at the current level and a recursive call to `flood` initiates the process for the peak component P_n^l rooted in n. During the recursive flooding, some points can be pushed in the queue between level λ and l. Hence, when `flood` ends, it returns the level l' of n's parent. If $l' > \lambda$, we need to flood levels l' until $l' \leq \lambda$ i.e. until there are no more points in the queue above λ. Once all pixels at level λ have been processed, we need to retrieve the level $lpar$ of the parent component and attach r to its canonical element. A *levroot* array stores the canonical element of each level component and -1 if the component is empty. Thus we just have to traverse *levroot* looking for $lpar = \max\{h < \lambda, levroot[h] \neq -1\}$ and set the parent of r to $levroot[lpar]$. Since the construction of *parent* is bottom-up, we can safely insert p in front of the S array each time $parent(p)$ is set. For a level component, the canonical element is the last element inserted ensuring a correct ordering of S. Note that the pass which gets the minimum level of the image is not necessary. Instead, we could have called `flood` in `Max-tree` procedure until the parent level returned by the function was -1, i.e the last flood call was processing the root. Anyway, this pass has other advantages for optimization that will be discussed in the implementation details section.

Salembier et al. [114]'s algorithm was rewritten in a non-recursive implementation in [56] and later by [96] and [133]. These algorithms differ in only two points. First, [133] uses a pass to retrieve the root before flooding to mimics the original recursive version while [96] does not. Second, priority queues in [96] use an unacknowledged implementation of

Algorithm 10: Salembier et al. [114] max-tree algorithm.

```
function flood(λ, r)
    while hqueue[λ] not empty do
        p ← pop(hqueue[λ]);
        parent(p) ← r;
        if p ≠ r then
            insert_front(S, p)
        foreach n ∈ N(p) s.t.
        parent(p) = −1 do
            l ← f(n);
            if levroot[l] = −1 then
                levroot[l] ← n
            push(hqueue[l], n);
            parent(n) ← INQUEUE;
            while l > λ do
                l ← flood(l, levroot[l])
    /* Attach to parent          */
    levroot[λ] ← −1;
    lpar ← λ − 1;
    while lpar ≥ 0 and
    levroot[lpar] = −1 do
        lpar ← lpar − 1
    if lpar ≠ −1 then
        parent(r) ← levroot[lpar]
    insert_front(S, r);
    return lpar

function Max-tree(f)
    foreach h do levroot[h] ← −1;
    foreach p do parent(p) ← −1;
    l_min ← min_p f(p);
    p_min ← arg min_p f(p);
    push(hqueue[l_min], p_min);
    levroot[lmin] ← p_min;
    flood(l_min, p_min);
```

Algorithm 11: Non-recursive max-tree algorithm [96, 133].

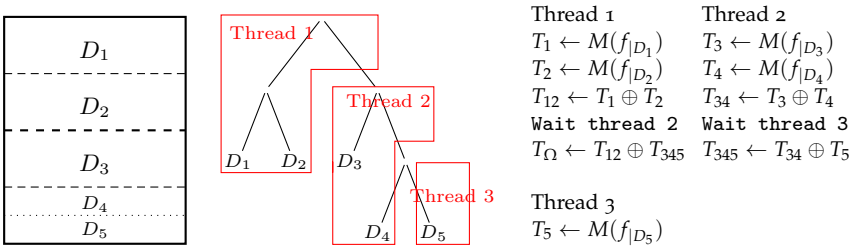
```
function ProcessStack(r, q)
    λ ← f(q);
    pop(levroot);
    while levroot not empty and
    λ < f(top(levroot)) do
        insert_front(S, r);
        r ← pop(levroot); parent(r) ← r;
    if levroot empty or f(top(levroot)) ≠ λ then
        push(levroot, q)
    parent(r) ← top(levroot);
    insert_front(S, r);

function Max-tree(f)
    foreach p do parent(p) ← −1;
    p_start ← any point in Ω;
    push(pqueue, p_start); push(levroot, p_start);
    parent(p_start) ← INQUEUE;
    loop
        p ← top(pqueue); r ← top(levroot);
        foreach n ∈ N(p) s.t. parent(p) = −1 do
            push(pqueue, n);
            parent(n) ← INQUEUE;
            if f(p) < f(n) then
                push(levroot, n);
                goto 16;
            /* p is done                  */
            pop(pqueue);
            parent(p) ← r;
            if p ≠ r then insert_front(S, p);
    while pqueue not empty
        // all pts at current level done?
        q ← top(pqueue);
        if f(q) ≠ f(r) then
            // Attach r to its parent
            ProcessStack(r, q);
    root ← pop(levroot);
    insert_front(S, root);
```

heap based on hierarchical queues while in [133] they are implemented using a standard heap (based on comparisons). The algorithm given in Algorithm 11 is a code transcription of the method described in [96]. The array *levroot* in the recursive version is replaced by a stack with the same purpose: storing the canonical element of level components.

The hierarchical queue *hqueue* is replaced by a priority queue *pqueue* that stores the propagation front. The algorithm starts with some initialization and chooses a random point p_{start} as the flooding point. p_{start} is enqueued and pushed on *levroot* as a canonical element. During the flooding, the algorithm picks the point p at highest level (with the highest priority) in the queue, and the canonical element r of its component which is the top of *levroot* (p is not removed from the queue). Like in the recursive version, we look for neighbors n of p and enqueue those that have not yet been seen. If $f(n) > f(p)$, n is pushed on the stack and we immediately flood n (a *goto* that mimics the recursive call). On the other hand, if all neighbors are in the queue or already processed then p is *done*, it is removed from the queue, $parent(p)$ is set its the canonical element r and if $r \neq p$, p is added to S (we have to ensure that the canonical element will be inserted last). Once p removed from the queue, we have to check if the level component has been fully processed in order to attach the canonical element r to its parent. If the next pixel q has a different level than p, we call the procedure `ProcessStack` that pops the stack, sets parent relationship between canonical elements and inserts them in S until the top component has a level no greater than $f(q)$. If the stack top's level matches q's level, q extends the component so that no more processing is needed. On the other hand, if the stack gets empty or the top level is lesser than $f(q)$, then q is pushed on the stack as the canonical element of a new component. The algorithm ends when all points in queue have been processed, then S only misses the root of the tree which is the single element that remains on the stack.

A.3.3 *Merge-based algorithms and parallelism*

Whereas the algorithms of the two first classes (Sections A.3.1 and A.3.2) are sequential, this section is dedicated to parallel algorithms. Merge-based algorithms consist in computing max-trees on sub-parts of images and merging back trees to get the max-tree of the whole image [100, 135, 77]. Those algorithms are typically well-suited for parallelism since they adopt a map-reduce idiom. Computation of sub max-trees (map step), done by any sequential method and merge (reduce-step) are executed in parallel by several threads. In order to improve cache coherence, images should be split in contiguous memory blocks that is, splitting along the first dimension if images are row-major. Figure 68 shows an example of parallel processing using a map-reduce idiom. The domain has been split into five sub-domains $\{D_1, D_2, \ldots, D_5\}$, we thus have 5 *map* operations which run a sequential algorithm and 4 *joins* that merge the sub-trees. Figures 68b and 68c show a possible distribution of the tasks on 3 threads. Note that *map*-steps and *reduce*-steps may occur in parallel, but a single thread may also be in charge of several sub-tree construction. For instance, the first thread is in charge of computing the sub-trees T_1 and T_2 for D_1 and D_2, merging them into a tree T_{12} and then merging it with the tree computed by the second thread. Choosing the right number of splits and jobs distribution between threads is a difficult topic that depends on the architecture (number of threads available, power frequency of each core). If the domain is not split enough (a number of chunks no greater



(a) (b) (c)

Figure 68: Map-reduce idiom for max-tree computation. (a) Sub-domains of f. (b) A possible distribution of jobs by threads. (c) Map-reduce operations. M is the map operator, $\oplus$ the merge operator.

Algorithm 12: Tree merge algorithm.

```
function connect(p,q)
    x ← findrepr(parent, p);
    y ← findrepr(parent, q);
    if f(x) < f(y) then swap(x, y);
    while x ≠ y do
        /* common ancestor found?                    */
        parent(x) ← findrepr(parent, parent(x));
        z ← parent(x);
        if x = z then parent(x) ← y ; y ← x;
        else if f(z) ≥ f(y) then x ← z;
        else  parent(x) ← y; x ← y; y ← z;

function findrepr(par, p)
    if f(p) ≠ f(par(p)) then return p;
    par(p) ← findrepr(par, par(p));
    return par(p)

function mergetree(D_i, D_j)
    foreach p ∈ D_i do
        foreach q ∈ (N(p) ∩ D_j) do
            connect(p,q)
```

than the number of threads) the parallelism is not maximal, some threads become idle once they have done their jobs, or wait for other thread to merge. On the other hand, if the number of split gets too large, merging and thread synchronization cause significant overheads. Since work balancing and thread management are outside the current topic, they are delegated to high level parallelism libraries such as Intel Threading Building Blocks (TBB).

The procedure in charge of merging sub-trees T_i and T_j of two adjacent domains D_i and D_j is given in Algorithm 12. For two neighbors p and q in the junction of D_i, D_j, it connects components of p's branch in T_i to components of q's branch in T_j until a common ancestor is found. Let x and y be the canonical elements of the components to merge with $f(x) \geq f(y)$ (x is in the childhood to y) and z be the canonical element of the parent component of x. If x is the root of the sub-tree then it gets attached to y and the procedure ends. Otherwise, we traverse up the branch of x to find the component that will be attached to y that is the lowest node having a level greater than $f(y)$. Once found, x gets attached to y, and we now have to connect y to x's old parent. `findrepr(p)` is used to get the canonical element of p's component whenever the algorithm needs it.

Algorithm 13: Canonicalization and S computation algorithm.

```
function CanonicalizeRec(p)
    dejavu(p) = true;
    q ← parent(p);
    if not dejavu(q) then
        // Process parent before p
        CanonicalizeRec(q);
    if f(q) = f(parent(q)) then  // Canonicalize
        parent(p) ← parent(q);
    InsertBack(S, p);

function Post-process(parent, f)
    foreach p do dejavu(p) ← False;
    foreach p ∈ Ω s.t. not dejavu(p) do
        CanonicalizeRec(p);
    return (parent, S)
```

Once sub-trees have been computed and merged into a single tree, it does not hold canonical property (because non-canonical elements are not updated during merge). Also, the reduction step does not merge the S arrays corresponding to sub-trees (it would imply reordering S which is more costly than just recomputing it at the end). Algorithm 13 shows an algorithm that canonicalizes and reconstructs S array from *parent* image. It uses an auxiliary image *dejavu* to track nodes that have already been inserted in S. As opposed to other max-tree algorithms, construction of S and processing of nodes are top-down. For any points p, we traverse in a recursive way its path to the root to process its ancestors. When the recursive call returns, $parent(p)$ is already inserted in S and holds the canonical property, thus we can safely insert back p in S and canonicalize p as in Algorithm 7.

A.3.4 *Implementation details*

Algorithms have been implemented in pure C++ using STL implementation of some basic data structures (heaps, priority queues), the MILENA image processing library to provide fundamental image types and I/O functionality, and Intel TBB [109] for parallelism. Specific implementation optimizations are listed below:

Sort optimization. A counting sort is used when quantization is lower than 18 bits. For large integers of q bits, it switches to 2^{16}-based radix sort requiring $q/16$ counting sorts.

Pre-allocation. Queues and stacks are pre-allocated to avoid dynamic memory reallocation. Hierarchical queues are also pre-allocated by computing image histogram as a pre-processing.

Priority-queues. A heap is implemented with hierarchical queues when quantization is less than 18 bits. For large integer it switches to the STL standard heap implementation. A "y-fast trie" data structure [136] can be used for large integer ensuring a better complexity (see Appendix A.4.1) but no performance gain has been obtained.

Map-reduce. In the parallel version of the algorithms, all instructions that deal about S construction and canonicalization have been removed since S is reconstructed from scratch and *parent* canonicalized by the procedure in Algorithm 13

	Time complexity			Auxiliary space requirement		
Algorithm	Small int	Large int	Generic V	Small int	Large int	Generic V
Berger [15]	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$n + k + O(n)$	$2n + O(n)$	$n + O(n)$
Berger + rank	$O(n\,\alpha(n))$	$O(n \log\log n)$	$O(n \log n)$	$3n + k + O(n)$	$4n + O(n)$	$3n + O(n)$
Najman and Couprie [91]	$O(n\,\alpha(n))$	$O(n \log\log n)$	$O(n \log n)$	$5n + k + O(n)$	$6n + O(n)$	$5n + O(n)$
Salembier et al. [114]	$O(nk)$	$O(nk) \simeq O(n^2)$	N/A	$3k + n + O(n)$	$2k + n + O(n)$	N/A
Nistér and Stewénius [96]	$O(nk)$	$O(nk) \simeq O(n^2)$	N/A	$2k + 2n$	$2k + 2n$	N/A
Wilkinson [133]	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$3n$	$3n$	$3n$
Salembier non-recursive	$O(nk)$	$O(n \log\log n)$	$O(n \log n)$	$2k + 2n$	$3n$	$3n$
Map-reduce	$O(A(k,n))$	$O(A(k,n)) + O(k\sqrt{n} \log n)$		$\ldots + n$	$\ldots + n$	$\ldots + n$
Matas et al. [77]	$O(n)$	$O(n) + O(k\sqrt{n}(\log n)^2)$		$k + n$	$2n$	$2n$

Table 3: Complexity and space requirements of many max-tree algorithms. n is the number of pixels and k the number of gray levels.

A.4 ALGORITHMS COMPARISON

A.4.1 *Complexity analysis*

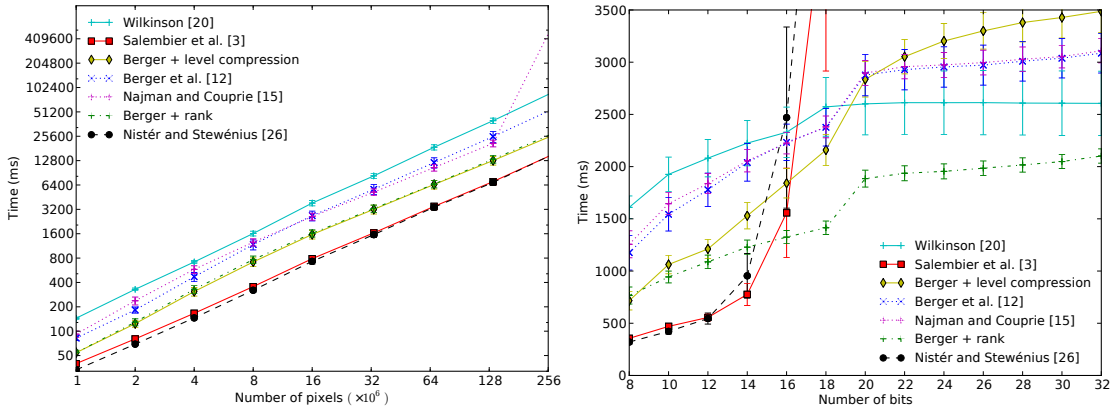
Let $n = H * W$ with H the image height, W the image width and n the total number of pixels. Let k be the number of values in V.

A.4.1.1 *Immersion algorithms*

They require sorting pixels, a process of $\Theta(n + k)$ complexity ($k \ll n$) for small integers (counting sort), $O(n \log\log n)$ for large integers (hybrid radix sort), and $O(n \log n)$ for generic data types with a more complicated ordering relation (comparison sort). Union-find is $O(n \log n)$ and $O(n\alpha(n))$ when used with union-by-rank [1]. Canonicalization is linear and does not use extra memory. Memory-wise, sorting may require an auxiliary buffer depending on the algorithm and histograms for integer sorts thus $\Theta(n + k)$ extra-space. Union without rank requires a *zpar* image for path compression ($\Theta(n)$) and the system stack for recursive calls in `findroot` which is $O(n)$ (`findroot` could be non-recursive, but memory space is saved at the cost of a higher computational time). Union-by-rank requires two extra images (*rank* and *repr*) of n pixels each.

A.4.1.2 *Flooding algorithms*

They require a priority queue to retrieve the highest point in the propagation front. Each point is inserted and removed once, thus the complexity is $\Theta(np)$ where p is the cost of pushing or popping from the heap. If the priority queue is encoded with a hierarchical queue as in [114, 96], it uses $n + 2k$ memory space, provides constant insertion and constant access to the maximum but popping is $O(k)$. In practice, in images with small integers, gray level difference between neighboring pixels is far to be as large as k. With high dynamic image, a heap can be implemented with a y-fast trie [136], which has insertion and deletion in $O(\log\log k)$ and access to maximum element in $O(1)$. For any other data type, a "standard" heap based on comparisons requires n extra space, allows



(a) (b)

Figure 69: (a) Comparison of the algorithms on 8-bit images as a function of the size; (b) Comparison of the algorithms on 8 Mega-pixels images as a function of the quantization.

insertion and deletion in $O(\log n)$ and has a constant access to its maximal element. Those algorithms need an array or a stack of respective size k and n. Salembier's algorithm uses the system stack for a recursion of maximum depth k, hence $O(k)$ extra-space.

A.4.1.3 *Merge-based algorithms*

The complexity depends on $\mathcal{A}(k, n)$, the complexity of the underlying method used to compute the max-trees of sub-domains. Let $s = 2^h$ the number of sub-domains. The map-reduce algorithms require s mapping operations and $s - 1$ merges. A good map-reduce algorithm would split the domain to form a full and complete tree so we assume all leaves to be at level h. Merging sub-trees of size $n/2$ has been analyzed in [135] and is $O(k \log n)$ (we merge nodes of every k levels using union-find without union-by-rank). Thus, the complexity of a single reduction is $O(Wk \log n)$. Assuming s constant and $H = W = \sqrt{n}$ the complexity as a function of n and k of the map-reduce algorithm is $O(\mathcal{A}(k, n)) + O(k\sqrt{n} \log n)$. When there is as many splits as rows, s is now dependent on n. This leads to Matas et al. [77] algorithm whose complexity is $O(n) + O(k\sqrt{n}(\log n)^2)$. Contrary to what they claim, when values are small integers the complexity stays linear and is not dominated by merging operations. Finally, canonicalization and S reconstruction have a linear time complexity (`CanonicalizeRec` is called only once for each point) and only use an image of n elements to track points already processed. The complexity analysis for each algorithm as well as the memory required by any auxiliary data structure (including preallocated stacks and queues) is summerized in Table 3.

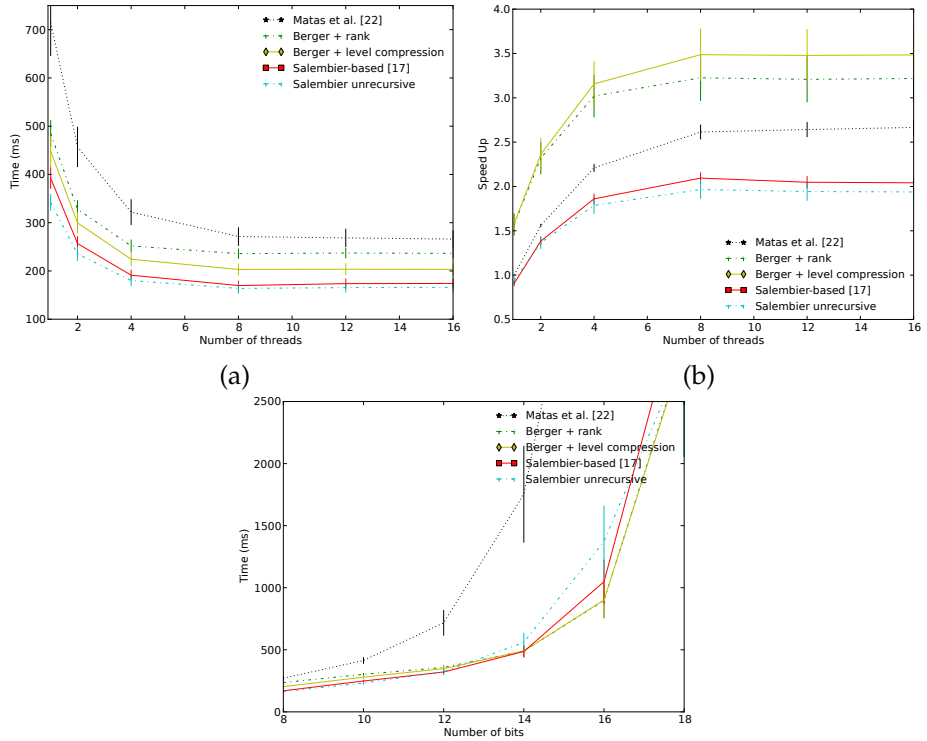


Figure 70: (a,b) Comparison of the parallel algorithms on a 6.8 Mega-pixels 8-bits image as a function of number of threads. (a) Wall clock time; (b) speedup w.r.t the sequential version; (c) Comparison of the parallel algorithms using 8 threads on a 6.8 Mega-pixels image as a function of the quantization.

A.4.2 *Experiments*

Benchmarks were performed on an Intel Core i7 (4 physical cores, 8 logical cores). The programs were compiled with gcc 4.7, optimization flags on (`-O3 -march=native`). Tests were conducted on a dataset of 8-bit images that were re-sized by cropping or tiling the original image. Over-quantization was performed by shifting the eight bits left and generating missing lower bits at random. Figure 69 depicts performance of the sequential algorithms w.r.t to the size and the quantization. As a first remark, we notice that all algorithms are linear in practice. On natural images, the $n \log n$ upper bound complexity of the [133] and [15] algorithms is not reached. Algorithms from [15] and [91] have quite the same running time ($\pm 6\%$ on average), however the performance of [91] algorithm drops significantly at 256 Mega-pixels. Indeed, at that size each auxiliary array/image requires 1 GB memory space, thus [91], which use a lot of memory, exceeds the 6 GB RAM limit and needs to swap. Our implementation of union-by-rank uses less memory and is on average 42% faster than [91]. Level compression is an efficient optimization that provides 35% speedup on average on [15]. However, this optimization is only reliable

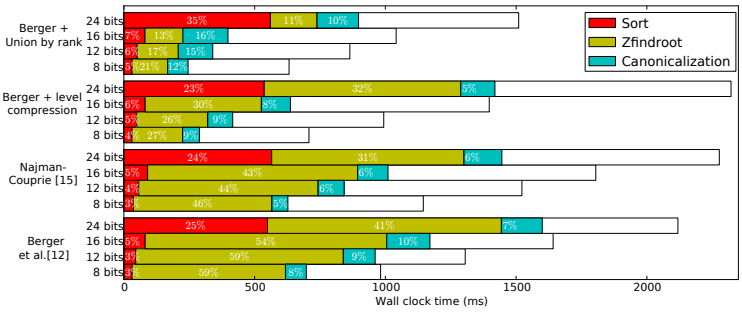


Figure 71: Time distribution of the sequential union-find-based algorithms.

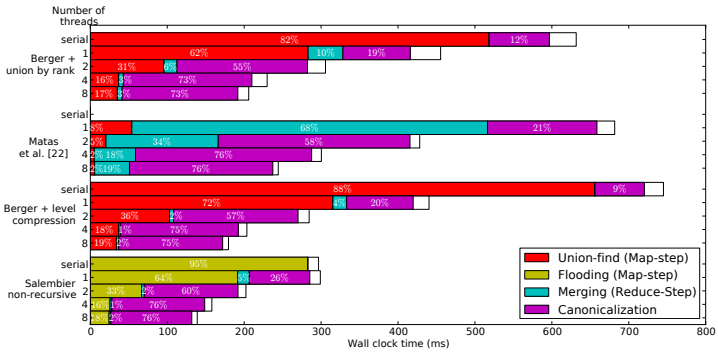


Figure 72: Time distribution of the parallel versions of the algorithms.

on low quantized data. Figure 69b shows that it is relevant up to 18 bits. It operates on flat-zones but when quantization gets higher, flat-zones are less probable and the tests add worthless overheads (see Figure 71). Union-find is not affected by the quantization but sorting does, counting sort and radix sort complexities are respectively linear and logarithmic with the number of bits. The break in union-find curves between 18 and 20 bits stands for the switch from counting to radix sort. Flooding-based algorithms using hierarchical queues outperform our union-find by rank on low quantized image by 41% on average. As expected, [114] and [96] (which is the exact non-recursive version of the former) closely match. However, the exponential cost of hierarchical queues w.r.t the number of bits is evident on Figure 69b. By using a standard heap instead of hierarchical queues, [133] does scale well with the number of bits and outperforms every algorithms except our implementation of union-by-rank. In [133], the algorithm is supposed to match [114]'s method for low quantized images, but in our experiments it remains 4 times slower. Since [91]'s algorithm is always outperformed by our implementation of union-find by rank, it will not be tested any further. Furthermore, because of the strong similarities of [96] and [133], they are merged in our single implementation (called *Non-recursive Salembier* below) that will use hierarchical queues when quantization is below 18 bits and switches to a standard heap implementation otherwise. Finally, the algorithm *Berger + level compression* will enable level compression only when the number of bits is below 18.

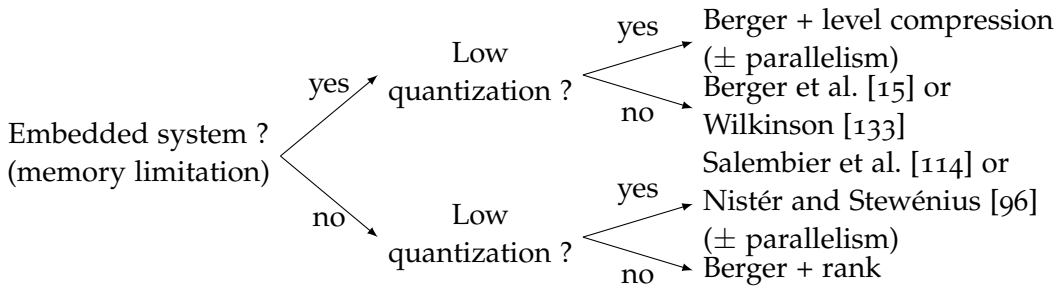


Figure 73: Decision tree to choose the appropriate max-tree algorithm.

Figure 70 shows the results of the map-reduce idiom applied on many algorithms and their parallel versions. As a first result, we can see that better performance is generally achieved with 8 threads that is when the number of threads matches the number of (logical) cores. However, since there are actually only 4 physical cores, we can expect a $\times 4$ maximum speedup. Some algorithms benefit more from map-reduce than others. Union-find-based algorithms are particularly well-suited for parallelism. Union-find with level compression achieves the best speedup, 3.6 times faster that the sequential version while the union-find by rank, second, performs a $\times 3.1$ speedup. More surprisingly, the map-reduce pattern achieves significant speedup even when a single thread is used ($\times 1.7$ and $\times 1.4$ for union-find with level compression and union-find by rank respectively). This result is explained by a better cache coherence when working on sub-domains that balances tree merges overheads. On the other hand, flooding algorithms do not scale that well because they are limited by post-processes. Indeed, Figure 72 shows that 76% of the time of parallelized Salembier's algorithm is spent in post-preprocessing (that is going to happen as well for union-find algorithms on architectures with more cores). In [135] and [77], they obtain a speedup almost linear with the number of threads because only a *parent* image is built. If we remove the canonicalization and the S construction steps, we also get those speedups. Figure 70c shows the exponential complexity of merging trees as number of bits increases that makes parallel algorithms unsuitable for high quantized data. In light of the previous analysis, Figure 73 provides some guidelines on how to choose the appropriate max-tree algorithm *w.r.t.* to image types and architectures.

A.5 CONCLUSION

In this paper, we tried to lead a fair comparison of max-tree algorithms in a unique framework. We highlighted the fact that there is no such thing as the "best" algorithm that outranks all the others in every case and we provided a decision tree to choose the appropriate algorithm *w.r.t.* to data and hardware. We proposed a max-tree algorithm using union-by-rank that outperforms the existing one from [91]. Furthermore, we proposed a second one that uses a new technique, *level compression*, for systems with strict memory constraints. Extra-materials including the image dataset used for this

comparison, and a "reproducible research" code, intensively tested, is available on the Internet at `http://www.lrde.epita.fr/Olena/MaxtreeReview`.

Actually the union-find algorithm is a versatile tool used in many algorithms. A recent publication [46] shows that the morphological *tree of shapes*, which is a self-dual representation of the image contents, can also be computed using union-find. In [95], a specific binary tree, corresponding to an ordered version of the edges of the minimum spanning tree, is computed thanks to a Kruskal-like algorithm and involves the union-find algorithm. Thus, the results presented in this paper also apply to obtain those trees in the most efficient way.

B

TREE OF SHAPES COMPUTATION ALGORITHM

Thierry Géraud, Edwin Carlinet, Sébastien Crozet, and Laurent Najman. "A Quasi-linear Algorithm to Compute the Tree of Shapes of n-D images." In: *Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM'13)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science. Springer, 2013, pp. 98–110.

B.1 INTRODUCTION

The Tree of Shapes (ToS) [85] is an important morphological structure that represents images in a self-dual way. Shortly put it can be seen as the result of merging the pair of dual component trees, min-tree and max-tree, into a single tree. Using the ToS has many advantages. Since it is self-dual, it makes no assumption about the contrast of objects (either light object over dark background or the contrary). We only have one structure that represents the image contents so we do not have to juggle with the couple of dual trees. It intrinsically eliminates the redundancy of information contained in those trees. Last, it encodes the spatial inclusion of connected components in gray-level images so it is complementary to some other representations that focus on component (or region) adjacency. As a consequence the ToS is not only an easy access to self-dual operators such as grain filters but it has many applications, as listed in [80] (pp. 15–17), and some very recent works illustrate several powerful perspectives offered by that tree (see [139, 141, 144], and their bibliography).

In the following we consider a nD digital image u as a function defined on a regular cubical grid (precisely, $u : \mathbb{Z}^n \to \mathbb{Z}$), and to properly deal with some subsets of $\mathbb{Z}^n$ and with their complementary, we consider the dual connectivities c_{2n} and c_{3^n-1}. For any $\lambda \in \mathbb{Z}$, the lower (strict) cuts[1] and upper (large) cuts of u are defined as $[\,u < \lambda\,] = \{\, x \in X \mid u(x) < \lambda \,\}$ and $[\,u \geq \lambda\,] = \{\, x \in X \mid u(x) \geq \lambda \,\}$. From them we deduce two sets, $\mathcal{T}_<(u)$ and $\mathcal{T}_\geq(u)$, composed of the connected components of respectively lower and upper cuts of u: $\mathcal{T}_<(u) = \{\, \Gamma \in \mathcal{CC}_{c_{2n}}([\,u < \lambda\,]) \,\}_\lambda$ and $\mathcal{T}_\geq(u) = \{\, \Gamma \in \mathcal{CC}_{c_{3^n-1}}([\,u \geq \lambda\,]) \,\}_\lambda$, where $\mathcal{CC}$ denotes the operator that gives the set of connected components of a set. The elements of $\mathcal{T}_<(u)$ and $\mathcal{T}_\geq(u)$ respectively give rise to two dual trees: the min-tree and the max-tree of u. We then define two other sets, $\mathcal{S}_<(u)$ (set of lower shapes) and $\mathcal{S}_\geq(u)$ (set of upper shape), as the sets of components of resp. $\mathcal{T}_<(u)$ and

1 We can indifferently use the term "cut" or "threshold".

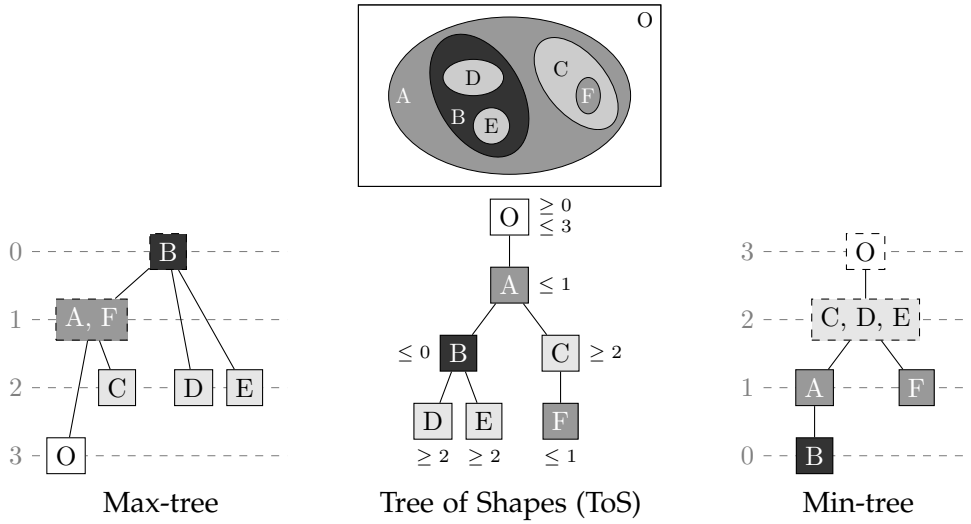


Figure 74: Three morphological trees of the same image.

$\mathcal{T}_{\geq}(u)$ after having filled the cavities [2] of those components. With the cavity-filling (or saturation) operator denoted by $\mathcal{H}$, we have: $\mathcal{S}_{<}(u) = \{\, \mathcal{H}_{c_{3^n-1}}(\Gamma);\ \Gamma \in \mathcal{T}_{<}(u) \,\}$ and $\mathcal{S}_{\geq}(u) = \{\, \mathcal{H}_{c_{2n}}(\Gamma);\ \Gamma \in \mathcal{T}_{\geq}(u) \,\}$.

The set of all shapes $\mathfrak{S}(u) = \mathcal{S}_{<}(u) \cup \mathcal{S}_{\geq}(u)$ forms a tree, the so-called Tree of Shapes (ToS) of u [85]. Indeed, for any pair of shapes Γ and Γ' in $\mathfrak{S}$, we have $\Gamma \subset \Gamma'$ or $\Gamma' \subset \Gamma$ or $\Gamma \cap \Gamma' = \emptyset$. Actually, the shapes are the cavities of the elements of $\mathcal{T}_{<}$ and $\mathcal{T}_{\geq}$. For instance, if we consider a lower component $\Gamma \in [\, u < \lambda \,]$ and a cavity H of Γ, this cavity is an upper shape, i.e., $H \in \mathcal{S}_{\geq}$. Furthermore, in a discrete setting, H is obtained after having filled the cavities of a component of $[\, u \geq \lambda \,]$. Figure 74 depicts on a sample image the three components trees ($\mathcal{T}_{<}$, $\mathcal{T}_{\geq}$, and $\mathfrak{S}$). Just note that the Equations so far rely on the pair of dual connectivities, c_{2n} and c_{3^n-1}, so discrete topological problems are avoided, and, in addition, we are forced to consider two kind of cuts: strict ones for c_{2n} and large ones for c_{3^n-1}.

The state-of-the-art of ToS computation (detailed in appendix B.5) suffers from two major flaws: existing algorithms have a time complexity of $O(n^2)$ and they cannot easily be extended to nD images. Briefly put, this is due to the fact that either they follow shape contours or they have to know if a component has a cavity[2]. This paper presents an algorithm that can compute the ToS with quasi-linear time complexity when image data are low quantized; furthermore this algorithm straightforwardly applies to nD images.

This paper is organized as follows. First we explain that a well-known algorithmic scheme can be reused to compute the ToS (appendix B.2). Then this paper introduces a new discrete representation of images (appendix B.3) that has some properties borrowed from the continuous world. At that point we are ready to glue together the algorithmic

2 In 2D, a cavity of a set $S \in \Omega$ is called a "hole"; in nD, it is a connected component of $\Omega \backslash S$ which is not the "exterior" of S. Browsing the elements of S in nD, with $n \geq 3$, does not allow to know whether S has a cavity or not[54].

scheme and the novel image representation to present a quasi-linear algorithm that compute the ToS (appendix B.4). Related works about that tree computation is presented so that the reader can compare our approach to existing ones (appendix B.5). Last, we give a short conclusion (appendix B.6) [3].

B.2 ALGORITHMIC SCHEME AND THE NEED FOR CONTINUITY

This section shows that the max-tree algorithm presented in [15] is actually an algorithmic "canvas" [47], that is, a kind of meta-algorithm that can be "filled in" so that it can serve different aims. In the present paper it gives an algorithm to compute the ToS.

B.2.1 *About union-find and component trees*

An extremely simple union-find structure (attributed by Aho to McIlroy and Morris) was shown by Tarjan [122] to be very efficient. This structure, also called disjoint-set data structure or merge-find set, has many advantages that are detailed in [27]; amongst them, memory compactedness, simplicity of use, and versatility. This structure and its related algorithms are of prime importance to handle connected operators [79, 45].

Algorithm 14: "Union-Find"-based computation of a morphological tree.

```
function unionfind(R)
    for all p do
        zpar(p) ← undef
    for i ← N − 1 to o do
        p ← R[i]
        parent(p) ← p
        zpar(p) ← p
        for all n ∈ N(p) s.t.
        zpar(n) ≠ undef do
            r ← findroot(zpar, n)
            if r ≠ p then
                parent(r) ← p
                zpar(r) ← p
    return parent

function findroot(zpar, x)
    if zpar (x) = x then
        return x
    else
        zpar(x) ← findroot(zpar, zpar(x))
        return zpar(x)

function computetree(u)
    R ← SORT(u)
    parent ← unionfind(R)
    canonicalizetree(u, R, parent)
    return (R, parent)
```

3 Due to limited place, this paper does not contain the following topics (they will be included into an extended version of this paper). *A comparison of execution times of existing algorithms.* Actually it is possible to reduce the space complexity (i.e., memory usage) of the algorithm proposed in this paper so the shorter version presented here is not our "competitive" version. *The union-by-rank procedure that guaranties quasi-linear complexity.* So that the UNION-FIND routine (given in [15] and recalled in algorithm 14) remains short, its code does not feature tree balancing; yet it is explained in [27]. *A formal proof of our algorithm.* This paper focuses on how the proposed algorithm works and gives an insight into the reasons why it works; to give a formal proof requires a large amount of materials, the first part of which can be found in [94]. *About high bit-depths data.* That case is not detailled in this paper.

Let us denote by $\mathcal{R}$ the ancestor relationship in trees: we have $a\,\mathcal{R}\,p$ iff a is an ancestor of p. $\mathcal{R}$ can be encoded as an array of elements (nodes) so that $a\,\mathcal{R}\,p \Leftrightarrow index_{\mathcal{R}}(a) < index_{\mathcal{R}}(p)$; browsing that array thus corresponds to a downwards browsing of the tree, i.e., from root to leaves. To construct the max-tree of a given image, we rely on a rooted tree defined by a parenthood function, named *parent*, and encoded as an nD image (so $parent(p)$ is an nD point). When a node of the max-tree contains several points, we choose its first point (with respect to $\mathcal{R}$) as the representative for this node; that point is called a component "canonical point" or a "level root". Let Γ denote a component corresponding to a node of the max-tree, p_Γ its canonical element, and p_r the root canonical element. The *parent* function that we want to construct should verify the following four properties: **1.** $parent(p_r) = p_r$; **2.** $\forall\, p \neq p_r,\ parent(p)\,\mathcal{R}\,p$; **3.** p is a canonical element iff $p = p_r \vee u(parent(p)) \neq u(p)$; **4.** $\forall\, p,\ p \in \Gamma \Leftrightarrow u(p) = u(p_\Gamma) \wedge \exists\, i, parent^i(p) = p_\Gamma$ (therefore $\forall\, p \in \Gamma,\ p = p_\Gamma \vee p_\Gamma\,\mathcal{R}\,p$).

The routine UNION_FIND, given in algorithm 14, is the classical "union-find" algorithm [122] but *modified* so that it computes the expected morphological tree [15] while browsing pixels following $\mathcal{R}^{-1}$, i.e., from leaves to root (let us recall that we do not feature here the union-by-rank version). Its result is a *parent* function that fulfills those first four properties. Obtaining the following extra property, "5. $\forall p$, $parent(p)$ is a canonical element," is extremely interesting since it ensures that the parent function, when restricted to canonical elements only, gives a "compact" morphological tree such as the ones depicted in fig. 74. Precisely it allows to browse components while discarding their contents: a traversal is thus limited to one element (one pixel) per component, instead of passing through every image elements (pixels). Transforming the parent function so that property 5 is verified can be performed by a simple post-processing of the union-find computation. The resulting tree has now the simplest form that we can expect; furthermore we have an isomorphism between images and their canonical representations.

B.2.2 *Computing the max-tree and the ToS*

The algorithm presented in [15] to compute the max-tree is surprisingly also able to compute the ToS. The skeleton, or *canvas*, of this algorithm is the routine COMPUTE_TREE given in the right part of algorithm 14; it is composed of *three* steps: sort the image elements (pixels); then run the modified union-find algorithm to compute a tree encoded by a parent function; last modify the parent function to give that tree its canonical form.

In the case of the max-tree, the sorting step provides $\mathcal{R}$ encoded as an array of points sorted by increasing gray-levels in u, i.e., such that the array indices satisfy $i < i' \Rightarrow u(\mathcal{R}[i]) \leq u(\mathcal{R}[i'])$. When image data are low quantized, typically 12 bit data or less, then sorting points can be performed by a distribution sort algorithm. Last, the canonicalization post-processing is a trivial 5-line routine that the reader can find in [15]. In the case of the ToS, it is also a tree that represents an inclusion relationship between connected components of the input image. As a consequence a first important idea to

catch is that the ToS can be computed with the exact same routine, UNION_FIND, as the one used by max-tree.

B.2.3 *What if...*

The major and crucial difference between the max-tree and the ToS computations is obviously the sorting step. For the UNION_FIND routine to be able to compute the ToS using $\mathcal{R}^{-1}$, the SORT routine has to sort the image elements so that $\mathcal{R}$ corresponds to a downward browsing of the ToS. Schematically we expect that $\mathcal{R}$ contains the image pixels going from the "external" shapes to the "internal" ones (included in the former ones).

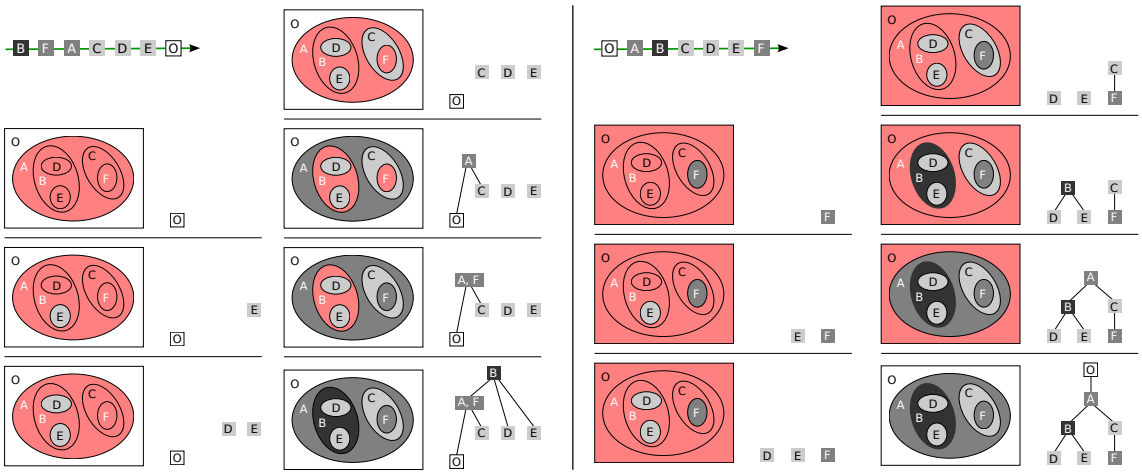


Figure 75: Tree computation of the max-tree (left) and of the ToS(right). For both cases, the result $\mathcal{R}$ of the sorting step is given over the green arrow and the tree computation, browsing $\mathcal{R}^{-1}$, is progressively depicted.

The similarity between the computations of both trees is illustrated in fig. 75. We can see that the modified union-find algorithm correctly computes both trees once $\mathcal{R}$ is properly defined. Therefore we "just" need to know how to compute $\mathcal{R}$ in the case of the ToS to turn the canvas given in the previous appendix B.2.2 as the expected algorithm.

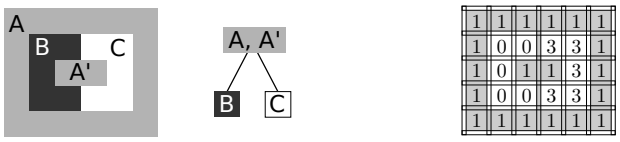


Figure 76: A sample image and its ToS (left); a step towards an *ad-hoc* image representation (right).

Let us consider the image depicted on the left of fig. 76 with its ToS. We can see that we need to reach the regions A and A' before the regions B and C in order to properly sort pixels, i.e., to compute $\mathcal{R}$. It is only possible if we can pass "between" pixels. The representation depicted on the right of fig. 76 is well-suited for that since it contains some elements that materialize inter-pixel spaces. Furthermore, given a two adjacent pixels with respective values 0 and 3, the element in-between them has to bear all the "intermediate" values: not only 1 but also 2. Indeed, if we change the value of regions A and A' from 1 to 2, the tree structure is unchanged but inter-pixel elements between regions B and C have now to make A and A' connect with value 2. Eventually we need an image representation that is "continuous" in some way with respect to both the domain space and the value space.

B.3 IMAGE REPRESENTATION

To be able to sort the image pixels so that $\mathcal{R}$ corresponds to a top-down browsing of ToS elements, this paper introduces a novel representation of images[4]. It relies on a couple of theoretical tools briefly described hereafter[5].

B.3.1 *Cellular complex and Khalimsky grid*

From the sets $H_0^1 = \{\{a\}; a \in \mathbb{Z}\}$ and $H_1^1 = \{\{a, a+1\}; a \in \mathbb{Z}\}$, we can define $H^1 = H_0^1 \cup H_1^1$ and the set H^n as the n-ary Cartesian power of H^1. If an element $h \subset \mathbb{Z}^n$ is the Cartesian product of d elements of H_1^1 and $n-d$ elements of H_0^1, we say that h is a d-face of H^n and that d is the dimension of h. The set of all faces, H^n, is called the nD space of cubical complexes. Figure 77 depicts a set of faces $\{f, g, h\} \subset H^2$ where $f = \{0\} \times \{1\}$, $g = \{0, 1\} \times \{0, 1\}$, and $h = \{1\} \times \{0, 1\}$; the dimension of those faces are respectively 0, 2, and 1. Let us write $h^{\uparrow} = \{h' \in H^n \mid h \subseteq h'\}$ and $h^{\downarrow} = \{h' \in H^n \mid h' \subseteq h\}$. The pair $(H^n, \subseteq)$ forms a poset and the set $\mathcal{U} = \{U \subseteq H^n \mid \forall h \in U, h^{\uparrow} \subseteq U\}$ is a To-Alexandroff topology on H^n. With $E \subseteq H^n$, we have a star operator $st(E) = \cup_{h \in E} h^{\uparrow}$ and a closure operator $cl(E) = \cup_{h \in E} h^{\downarrow}$, that respectively gives the smallest open set and the smallest closed set of $\mathcal{P}(H^n)$ containing E.

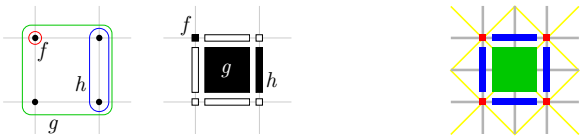


Figure 77: Three faces depicted as subsets of $\mathbb{Z}^2$ (left) and as geometrical objects (middle); Khalimsky grid (right) with 0- to 2-faces respectively painted in red, blue, and green.

4 In [94], a formal characterization of the discrete topology underlying this novel representation is presented.
5 The authors recommend [78, 11] for extra readings about those tools.

The set of faces of H^n is arranged onto a grid, the so-called Khalimsky's grid, depicted in gray in fig. 77 (right); and inclusion between faces lead to a neighborhood relationship, depicted in gray and yellow. The set of 2-faces, the minimal open sets of H^n, is the n-Cartesian product of H_1 and is denoted by H_1^n.

B.3.2 *Set-valued maps*

A set-valued map $\mathrm{U} : X \rightsquigarrow Y$ is characterized by its graph, $\mathrm{Gra}(\mathrm{U}) = \{ (x, y) \in X \times Y \,|\, y \in \mathrm{U}(x) \}$. There are two different ways to define the "inverse" of a subset by a set-valued map: $\mathrm{U}^{\oplus}(M) = \{ x \in X \,|\, \mathrm{U}(x) \cap M \neq \emptyset \}$ is the *inverse image* of M by U, whereas $\mathrm{U}^{\ominus}(M) = \{ x \in X \,|\, \mathrm{U}(x) \subset M \}$ is the *core* of M by U. Two distinct continuities are defined on set-valued maps. The one we are interested in is the "natural" extension of the continuity of a single-valued function. When X and Y are metric spaces and when $\mathrm{U}(x)$ is compact, U is said to be upper semi-continuous (u.s.c.) at x if $\forall \varepsilon > 0$, $\exists \eta > 0$ such that $\forall x' \in B_X(x, \eta)$, $\mathrm{U}(x') \subset B_Y(\mathrm{U}(x), \varepsilon)$, where $B_X(x, \eta)$ denotes the ball of X of radius η centered at x. One characterization of u.s.c. maps is the following: U is u.s.c. if and only if the core of any open subset is open.

B.3.3 *Interpolation*

Following the conclusions of appendix B.2.3, we are going to immerse a discrete nD function defined on a cubical grid $u : \mathbb{Z}^n \to \mathbb{Z}$ into some larger spaces in order to get some continuity properties. For the *domain space*, we use the subdivision $X = \frac{1}{2}H^n$ of H^n. Every element $z \in \mathbb{Z}^n$ is mapped to an element $m(z) \in \frac{1}{2}H_1^n$ with $z = (z_1, \ldots, z_n) \longmapsto m(z) = \{z_1, z_1 + \frac{1}{2}\} \times \ldots \times \{z_n, z_n + \frac{1}{2}\}$. The definition domain of u, $\mathcal{D} \subseteq \mathbb{Z}^n$, has thus a counterpart in X, that will also be denoted $\mathcal{D}$, and that is depicted in bold in fig. 78. For the *value space*, we immerse $\mathbb{Z}$ (the set of pixel values) into the larger space $Y = \frac{1}{2}H^1$, where every integer becomes a closed singleton of H_0^1. Thanks to an "interpolation" function, we can now define from u a set-valued map $\mathrm{U} = \mathcal{I}(u)$. We have $\mathrm{U} : X \rightsquigarrow Y$ and we set:

$$\forall\, h \in X,\ \ \mathrm{U}(h) = \begin{cases} \{\, u(m^{-1}(h)) \,\} & \text{if } \ h \in \mathcal{D} \\ \max(\, \mathrm{U}(h') : h' \in st(cl(h)) \cap \mathcal{D} \,) & \text{if } \ h \in \frac{1}{2}H_1^n \backslash \mathcal{D} \\ \mathrm{span}(\, \mathrm{U}(h') : h' \in st(h) \cap \mathcal{D} \,) & \text{if } \ h \in X \backslash \frac{1}{2}H_1^n. \end{cases} \tag{B.1}$$

An example of interpolation is given in fig. 78. Actually, whatever u, such a discrete interpolation $\mathcal{I}(u)$ can also be interpreted as a non-discrete set-valued map $\mathcal{I}_{\mathbb{R}}(u) : \mathbb{R}^n \rightsquigarrow \mathbb{R}$ (schematically $\mathcal{I}_{\mathbb{R}}(u)(x) = \mathcal{I}(u)(h)$ with h such as $x \in \mathbb{R}^n$ falls in $h \in \frac{1}{2}H^n$), and we can show that $\mathcal{I}_{\mathbb{R}}(u)$ is an u.s.c. map.

To the authors knowledge the notion of cuts (or thresholds) have not been defined for set-valued maps. Since they are of prime importance for mathematical morphology, and for the ToS in particular, we propose in this paper the following definitions. Given $\lambda \in Y$, let us state that $[\, \mathrm{U} \lhd \lambda \,] = \{ x \in X \mid \forall \mu \in \mathrm{U}(x),\ \mu < \lambda \}$ and $[\, \mathrm{U} \rhd \lambda \,] = \{ x \in$

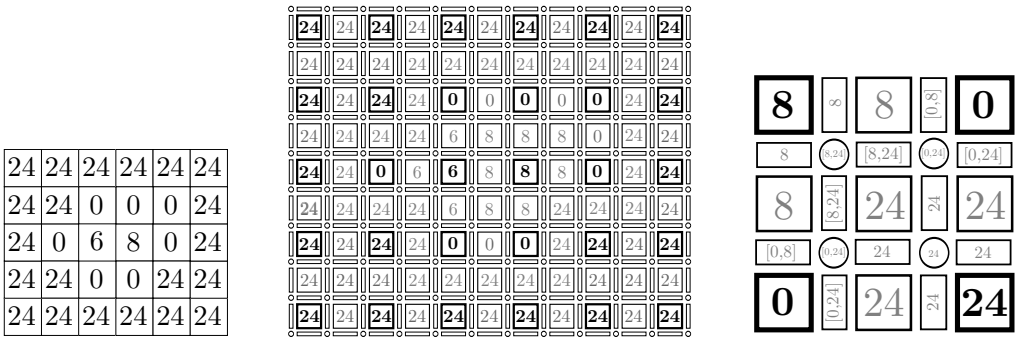


Figure 78: The function $u : \mathbb{Z}^2 \to \mathbb{Z}$ (left) is transformed into the set-valued map $\mathrm{U} : \frac{1}{2}H^2 \rightsquigarrow \frac{1}{2}H^1$ (middle); d-faces with $d \in \{0, 1\}$ are interval-valued in U with the span of their respective $(d+1)$-face neighbors (right).

$X \mid \forall\, \mu \in \mathrm{U}(x),\ \mu > \lambda\,\}$. We can show [94] that, with those definitions, $\forall u$, $\forall \lambda$, $[\,\mathfrak{I}(u) \lhd \lambda\,]$ and $[\,\mathfrak{I}(u) \rhd \lambda\,]$ are well-composed [65]. That is, strict cut components and their complementary sets can be handled both with the same unique connectivity, c_{2n}. As a consequence, the operators *star* and $\mathcal{H}$ commute on those sets, and we can prove [94] that:

$$\mathfrak{S}_{\mathfrak{I}}(u) \;=\; \{\, \mathcal{H}_{c_{2n}}(\Gamma);\ \Gamma \in \{\mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \lhd \lambda\,])\}_{\lambda} \cup \{\mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \rhd \lambda\,])\}_{\lambda}\,\}$$

is a set of components that forms a tree. Moreover, we can also prove that $\mathcal{T}_{<}(u) = \{\,\Gamma \cap \mathcal{D};\ \Gamma \in \{\,\mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \lhd \lambda + 1/2\,])\,\}_{\lambda \in H_0}\,\}$ and $\mathcal{T}_{\geq}(u) = \{\,\Gamma \cap \mathcal{D};\ \Gamma \in \{\,\mathcal{CC}_{c_{2n}}([\,\mathfrak{I}(u) \rhd \lambda + 1/2\,])\,\}_{\lambda \in H_0}\,\}$. So eventually we have: $\mathfrak{S}(u) = \{\,\Gamma \cap \mathcal{D};\ \Gamma \in \mathfrak{S}_{\mathfrak{I}}(u)\}$. That final property means that strict cuts of the interpolation of u, considering only c_{2n} for the different operators, allows for retrieving the shapes of u, as defined with the pair of dual connectivities c_{2n} and c_{3^n-1}.

B.4 PUTTING THINGS ALTOGETHER

B.4.1 *About saturation and initialization*

Classically the root node of the ToS represents the whole image and, formally, the saturation operator is defined w.r.t. a point at infinity, p_∞, located outside the image domain $\mathcal{D}$. A rather natural idea is that the root level, ℓ_∞, should only depend on the internal border of $\mathcal{D}$ (which is unfortunately not the case for the algorithms proposed in the literature). To that aim, before interpolating u, we add to this image an external border with a unique value, ℓ_∞, set to the median value of the internal border. p_∞ is then one point from the added border.

Algorithm 15: Sorting for ToS computation.

```
function priority_push(q, h, U, ℓ)
    // modify q
    [lower, upper] ← U(h)
    if lower > ℓ then ℓ' ← lower
    else if upper < ℓ then ℓ' ← upper
    else ℓ' ← ℓ
    PUSH(q[ℓ'], h)
function priority_pop(q, ℓ)
    // modify q, and sometimes ℓ
    ℓ' ← level next to ℓ such as q[ℓ'] is not empty
    ℓ ← ℓ'
    return POP(q[ℓ])
```

```
function sort(U)
    for all h do
        deja_vu(h) ← false
    i ← 0
    PUSH(q[ℓ∞], p∞)
    deja_vu(p∞) ← true
    ℓ ← ℓ∞  /* start from root level */
    while q is not empty do
        h ← priority_pop(q, ℓ)
        u^b(h) ← ℓ
        R[i] ← h
        for all n ∈ N(h) s.t. not deja_vu(n) do
            priority_push(q, n, U, ℓ)
            deja_vu(n) ← true
        i ← i + 1
    return (R, u^b)
```

B.4.2 *Handling the hierarchical queue*

To sort the faces of the domain X of U, we use a classical front propagation based on a hierarchical queue [83], denoted by q, the current level being denoted by ℓ. The sorting algorithm is given in algorithm 15. There are two notable differences with the well-known hierarchical-queue-based propagation. First the d-faces, with $d < n$, are interval-valued so we have to decide at which (single-valued) level to enqueue those elements. The solution is straightforward: a face h is enqueued at the value of the interval $U(h)$ that is the closest to ℓ (see the procedure `priority_push()`). Just also note that we memorize the enqueuing level of faces thanks to the image $u^\flat$ (see the procedure SORT). Second, when the queue at current level, $q[\ell]$, is empty (and when the hierarchical queue q is not yet empty), we shall decide what the next level to be processed is. We have the choice of taking the next level, either less or greater than ℓ, such that the queue at that level is not empty (see the procedure `priority_pop()`). Practically choosing going up or down the levels does not change the resulting tree since it just means exploring some sub-tree before some other disjoint sub-tree.

The result $\mathcal{R}$ of the sorting step is the one expected since the image U, in addition with the browsing of level in the hierarchical queue, allows for a propagation that is "continuous" both in domain space and in level space. An interesting property due to the interpolation and the well-composedness of cuts is that the neighborhood $\mathcal{N}$, used for faces in the propagation, corresponds to the c_{2n} connectivity on the Khalimsky's grid.

B.4.3 *Max-tree versus tree of shapes computation*

The main body of the ToS computation algorithm is given in algorithm 16. The major differences between this algorithm and the one dedicated to the max-tree (see the procedure COMPUTE_TREE in algorithm 14) are the following ones.

Algorithm 16: ToS computation in five steps.

```
function compute_tree_of_shapes(u)
    U ← interpolate(u)
    (R, u♭) ← sort(U)
    parent ← union_find(R)
    canonicalize_tree(u♭, R, parent)
    return un-interpolate(R, parent)
```

First the three basic steps (sort, union-find, and canonicalization) are now surrounded by an interpolation and un-interpolation process. Note that the un-interpolation just cleans up both $\mathcal{R}$ and *parent* to keep only elements of $\mathcal{D}$. Second, as emphasized in appendix B.2.2, the sorting step is of course dedicated to the ToS computation. Last, a temporary image, $u^\flat$, is introduced. It is defined on the same domain as U, namely X, and contains only single-valued elements. This image is the equivalent of the original image u when dealing with the max-tree: it is used to know when an element h is canonical, that is, when $u^\flat(parent(h)) \neq u^\flat(h)$ (so that image is thus required by the canonicalization step that runs on X).

Complexity analysis of the algorithm presented here is trivial. The interpolation, canonicalization, and un-interpolation are linear. The modified union-find (once augmented with tree balancing, i.e., union-by-rank) is quasi-linear when values of the input image u have a low quantization (typically 12 bits or less). Last, the time complexity of the sorting step is governed by the use hierarchical queue: it is linear with low quantized data[6]. Eventually we obtain a quasi-linear algorithm. The representation of the tree with the pair $(\mathcal{R}, parent)$ allows for any manipulation and processing that one expects from a morphological tree [27].

B.5 RELATED WORKS

The first known algorithm, the "Fast Level Line Transform (FLLT)" [85], computes the max-tree and the min-tree of an image and obtains the ToS by merging both trees. The main drawback of the FLLT is the need to know that a component has an hole (in order to match it with a component of the other tree). To that aim the Euler characteristic is computed, which can be done *locally* (while following the border of components) but

6 Formally the sorting step has the pseudo-polynomial $O(k\,n)$ complexity, k being the number of different gray values. Though, since we consider low bit-depths data, k shall only be considered as a complexity multiplicative factor.

in 2D only. In [30, 80] the authors show that this fusion approach is sound in nD with $n > 2$; yet it cannot be effective in practice due to unacceptable complexity.

In [28] the "Fast Level Set Transform" (FLST) relies on a region-growing approach to decompose the image into shapes. It extracts each branch of the tree starting from the leaves and growing them up to the root until at least one saddle point is encountered. Each time a saddle point is encountered, the branch extraction procedure has to stop until every parallel branch meeting at this point is extracted. So each saddle point invalidates the shape currently being extracted, forcing the algorithm to visit its pixels again once a parallel branch is extracted. Since an image like a checkerboard contains $O(n)$ saddle points meeting on $O(n)$ pixels, the FLST has a $O(n^2)$ worst case time complexity.

Song [119] takes a top-down approach to build the *tree of level lines* in $O(n + t)$ time, where t is the total length of all level lines (note that filling the interior of each level line allows for retrieving the ToS). The algorithm is restricted to 2D images with hexagonal pixels. Its key idea is to perform a recursion (starting from the image boundary): for a given component, follow every contours of its holes, and repeat this procedure for each hole component. Since the total length of level lines of an image can be of order $O(n^2)$, the worst case has a quadratic-time complexity.

B.6 CONCLUSION

In this paper, we have presented a new algorithm to compute the ToS of an image which features a quasi-linear time complexity, runs on nD images, and benefits from a much simpler implementation than existing algorithms. We have also proposed a novel representation of images as set-valued maps which has some continuity properties while remaining discrete.

Actually we believe that this representation is a good start to get a *"pure" self-duality* for images and operators, that is, a way to get rid of the pair of dual connectivities c_{2n} and c_{3^n-1}, and of the dissymmetry of cuts (strict and large cuts for respectively lower and upper cuts). In particular, replacing the maximum operator by the median operator in eq. (B.1) leads to a pure self-dual definition of the ToS of 2D images [94]. Furthermore the perspectives offered by that new representation might be far from being limited to the ToS computation.

For our experiments we use our free software library [69]; in particular, the fact that our tool makes it easy to write generic software in the case of mathematical morphology and discrete topology is discussed in [70]. The work presented here will be available in the next release of our software for we advocate reproducible research.

Acknowledgements. The authors would like to thanks Michel Couprie and Jean Cousty for fruitful discussions. This work received funding from the Agence Nationale de la Recherche, contract ANR-2010-BLAN-0205-03 and through "Programme d'Investissements d'Avenir" (LabEx BEZOUT n°ANR-10-LABX-58).

C

EFFICIENT COMPUTATION OF ATTRIBUTES AND SALIENCY MAPS

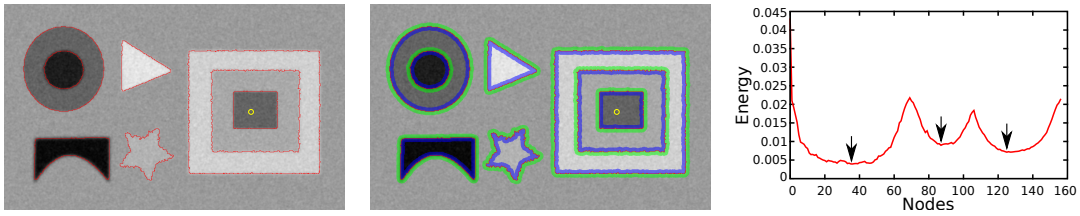
Yongchao Xu, Edwin Carlinet, Thierry Géraud, and Laurent Najman. "Efficient Computation of Attributes and Saliency Maps on Tree-Based Image Representations". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science Series. Reykjavik, Iceland: Springer, 2015, pp. 693–704.

C.1 INTRODUCTION

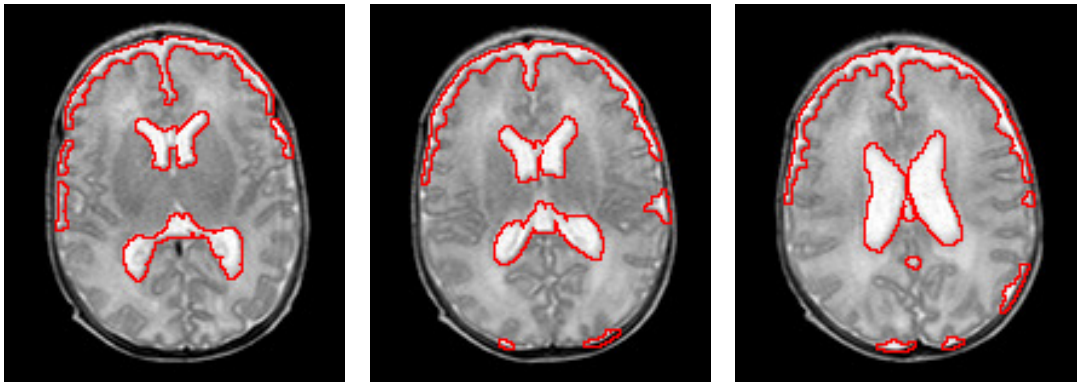
In a large number of applications, processing relies on objects or areas of interest. Therefore, region-based image representations have received much attention. In mathematical morphology, several region-based image representations have been popularized by attribute filters [18, 132] or connected operators [114, 113], which are filtering tools that act by merging flat zones. Such operators rely on transforming an image into an equivalent region-based representation, generally a tree of components (*e.g.*, the Min/Max-trees [114] or the tree of shapes [85]). Such trees are equivalent to the original image in the sense that the image can be reconstructed from the associated tree. Filtering then involves the design of an attribute function that weighs how important/meaningful a node of the tree is or how much a node of the tree fits a given shape. The filtering is achieved by preserving and removing some nodes of the tree according to the attribute function. This filtering process is either performed classically by thresholding the attribute function [113] or by considering the tree-based image representations as graphs and applying some filters on this graph representation [141, 137].

There exist many applications in image processing and computer vision that rely on tree-based image representations (see [137] for a short review). All these applications share a common scheme: one computes a tree representation and an attribute function upon which the tree analysis is performed. The choice of tree representation and the adequacy of attribute function mainly determine the success of the corresponding applications.

Many algorithms for computing different trees have been proposed (see appendix C.2.2 for a short review). In this paper, we focus on attribute computation, which is also an important step for the tree-based applications. To the best of our knowledge, only the algorithms for information computed on region have been presented [135] so far,



(a) Illustration on a synthetical image. Green: exterior region; Blue: interior region.



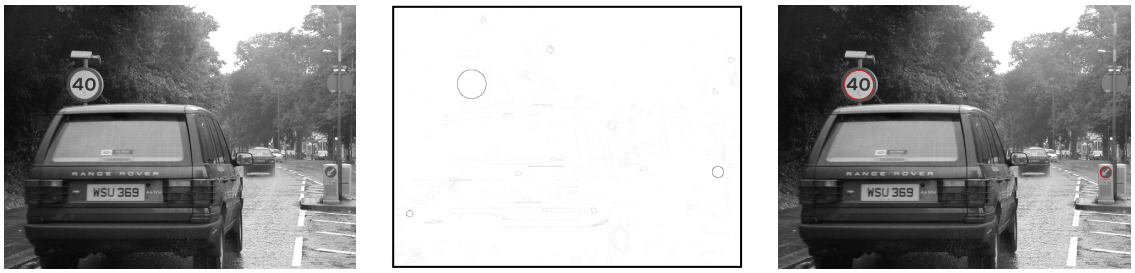
(b) Illustration of cerebrospinal fluid detection on MRI images of a newborn's brain.

Figure 79: Examples of object detection using the context-based energy estimator [139] relying on contour and context information. An evolution of this attribute along a branch starting from the yellow point to the root is depicted on the right side of (a).

none of the existing papers gives explicitly the algorithms computing the other attribute information employed in tree-based applications. In this paper, firstly, we detail explicitly how to incrementally compute some information on region, contour, and context. These informations form the basis for many classical attribute functions (*e.g.*, area, compactness, elongation). Let us remark that contextual information is very adequate for object detection, such as the context-based energy estimator [139] that relies on information computed on contour and context. Two examples of object detection using this attribute are shown in fig. 79. Another type of interesting information is extremal information along the contour (*e.g.*, the minimal gradient's magnitude along the boundary). An example employing this information is the number of false alarms (NFA) for meaningful level lines extraction [42, 21]. Here we propose an efficient algorithm that does not require much memory to compute this kind of information. Lastly, we depict an algorithm computing the extinction-based saliency map [137] representing a hierarchical morphological segmentation using tree-based image representations (two examples are illustrated in fig. 80). These algorithms form the main contribution of this paper.

The rest of the paper is organized as follows: A short review of some tree-based image representations and their computations using immersion algorithm are provided in appendix C.2. Our proposed algorithms to compute some attribute information

(a) Extinction-based saliency map using color tree of shapes [25].



(b) Circular object oriented extinction-based saliency map.

Figure 80: Illustrations of extinction-based saliency maps from the ToS.

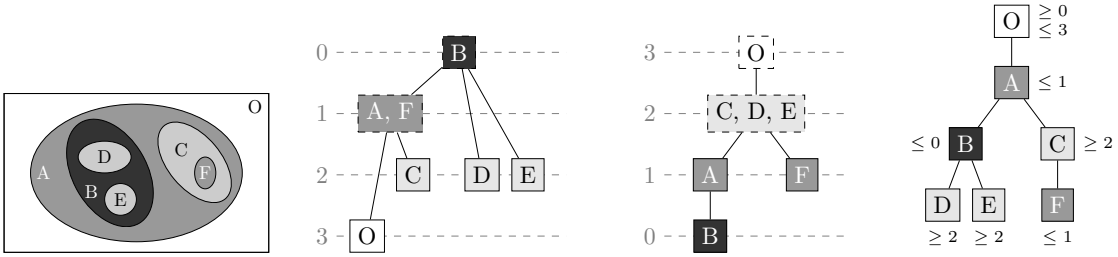
and saliency maps are detailed in appendix C.3, and we analyze in appendix C.4 the complexity and the memory cost of the proposed algorithms. Finally, we conclude and give some perspectives in appendix C.5.

C.2 REVIEW OF MORPHOLOGICAL TREES AND THEIR COMPUTATIONS

Region-based image representations are composed of a set of regions of the original image. Those regions are either disjoint or nested, and they are organized into a tree structure thanks to the inclusion relationship. There are two types of such representations: fine to coarse hierarchical segmentations and threshold decomposition-based trees. In this paper, we only consider the threshold decomposition-based trees.

C.2.1 *Tree-based Image Representations*

Let f be an image defined on domain Ω and with values on ordered set V (typically $\mathbb{R}$ or $\mathbb{Z}$). For any $\lambda \in V$, the upper level sets $\mathcal{X}_\lambda$ and lower level sets $\mathcal{X}^\lambda$ of an image f are respectively defined by $\mathcal{X}_\lambda(f) = \{p \in \Omega \mid f(p) \geq \lambda\}$ and $\mathcal{X}^\lambda(f) = \{p \in \Omega \mid f(p) \leq \lambda\}$. Both the upper and lower level sets have a natural inclusion structure: $\forall\, \lambda_1 \leq \lambda_2$, $\mathcal{X}_{\lambda_1} \supseteq \mathcal{X}_{\lambda_2}$ and $\mathcal{X}^{\lambda_1} \subseteq \mathcal{X}^{\lambda_2}$, which leads to two distinct and dual representations of the image: Max-tree and Min-tree [114]. The Tree of Shapes (ToS) is a fusion of the Max-tree and Min-tree via the notion of *shapes* [85]. A shape is defined as a connected component of an upper or lower level set with its holes filled in. Thanks to the inclusion relationship of both kinds of level sets, the set of shapes can be structured into a tree structure, called the ToS. An example of these trees is depicted in fig. 81.



(a) Input image. (b) Max-tree. (c) Min-tree. (d) Tree of shapes.

Figure 81: Tree-based image representations relying on threshold decompositions.

C.2.2 *Tree Computation and Representation*

There exist three types of algorithms to compute the Min/Max-tree (see [26] for a complete review): flooding algorithms [114, 133, 96], merge-based algorithms [135, 100], and immersion algorithms [15, 91]. In this paper, we employ the immersion algorithm to construct the Min/Max-tree. Concerning the ToS [85], there are four different algorithms [85, 119, 28, 46]. We use the one proposed by Géraud et al. [46]. It is similar to the immersion algorithms used for the Min/Max-tree computation. All these trees feature a common scheme of process: they start with considering each pixel as a singleton and sorting the pixels in decreasing tree order (*i.e.*, root to leaves order), followed by an union-find process (in reverse order) to merge disjoint sets to form a tree structure.

Let $\mathcal{R}$ be the vector of the N sorted pixels, and $\mathcal{N}(p)$ be neighbors (*e.g.*, 4- or 8-connectivity) of the pixel p. The union-find process is then depicted in the left column of algorithm 17, where *parent* and *zpar* are respectively the parenthood image and the root path compression image. The whole process of tree computation is given in the right column of algorithm 17, where SORT_PIXELS is a decreasing tree order sorting. The algorithms for computing the Min/Max-tree and the ToS differ in this pixel sorting step. For the Min/Max-tree, they are either sorted in decreasing order (Min-tree) or increasing order (Max-tree). If the image f is low quantized, we can use the Bucket sort algorithm to sort the pixels. Concerning the ToS, the sorting step is more complicated. It first interpolates the scalar image to an image of range using a simplicial version of the 2D discrete grid: the Khalimsky grid as shown in fig. 82. We note $\mathcal{K}_\Omega$, the domain Ω immersed on this grid. In fig. 82a, the original points of the image are the 2-faces, the boundaries are materialized with 0-faces and 1-faces. The algorithm in [46] ensures that shapes are open connected sets (*e.g.*, the purple shape in fig. 82a) and that shapes' borders are composed of 0-faces and 1-faces only (*e.g.*, the dark curve in fig. 82a). We refer the interested reader to the work of Géraud et al. [46] for more details on this pixel sorting step.

The tree structure is encoded through the image *parent* : $\Omega \to \Omega$ or $\mathcal{K}_\Omega \to \mathcal{K}_\Omega$ that states the parenthood relationship between nodes. In *parent*, a node is represented by a single pixel (a 2-face of the Khalimsky grid in the case of the ToS) called the canonical

Algorithm 17: Tree construction relying on union-find process.

```
function find_root(zpar, x)
    if zpar(x) ≠ x then
        zpar(x) ← find_root(zpar, zpar(x));
    return zpar(x)
function union_find(R)
    foreach p do zpar(p) ← undef;
    for i ← N − 1 to 0 do
        p ← R[i];
        parent(p) ← p; zpar(p) ← p;
        foreach n ∈ N(p) s.t.
        zpar(n) ≠ undef do
            r ← find_root(zpar, n);
            if r ≠ p then
                parent(r) ← p; zpar(r) ← p;
    return parent
function canonize_T(f, R, parent)
    for i ← 0 to N − 1 do
        p ← R[i];
        q ← parent(p);
        if f(parent(q)) = f(q) then
            parent(p) ← parent(q);
    return parent
function compute_tree(f)
    R ← sort_pixels(f);
    parent ← union_find(R);
    parent ← canonize_T(f, R, parent);
    return parent
```

element, and each non-canonical element is attached to the canonical element representing the node it belongs to. In the following, we denote by getCanonical : $\Omega \to \Omega$ or $\mathcal{K}_\Omega \to \mathcal{K}_\Omega$, the routine that returns the canonical element of each point in the image.

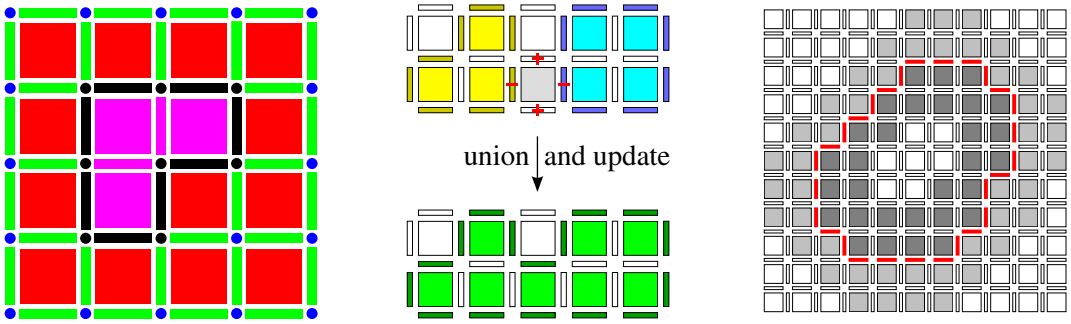
C.3 PROPOSED ALGORITHMS

In this section, we detail several algorithms related to some applications using tree-based image representations, including computation of some classical information used in many attribute functions (accumulated information in appendix C.3.1, and extremal information along the contour in appendix C.3.2), and computation of extinction-based saliency maps [137] in appendix C.3.3. For the sake of simplicity, we consider the Min-tree or Max-tree representation. The algorithms for the ToS construction share the same principle.

C.3.1 *Incremental Computation of Some Accumulated Information*

There are three main types of accumulated information: computed on region A (*e.g.*, area), on contour L (*e.g.*, length), and on context X (interior context X^i or exterior context X^e).

ATTRIBUTES COMPUTED ON REGIONS During the tree construction process, the algorithm starts with the pixels lying on the leaves, and the union-find acts as a region merging process. The connected components in the tree are built during this region growing process. We are able to handle information computed on region efficiently, such



(a) Khalimsky grid. (b) Updating contour. (c) Regional context.

Figure 82: (a): A point in a 2D image is materialized with 0-faces (blue disks), 1-faces (green strips), and 2-faces (red squares). (b): Updating contour information when an union between two components (yellow and blue) occurs thanks to a pixel (gray). (c): The approximated interior and exterior regional context of the red level line is respectively the dark gray region and the light gray region.

as its size, the sum of gray level or sum of square of gray level that can be used to compute the mean and the variance inside each region, the moments of each region based on which we can compute some shape attribute that measures how much a node fits a specific pattern. The algorithm for computing these information is depicted in algorithm 18 by adding some additional operations (red lines) to the union-find process during the tree construction, where i_A encodes information on pixels (*i.e.*, 2-faces). For example if A is the size or the sum of gray level, then i_A would be 1 (size of a pixel) or the pixel value. The operator $\widehat{+}$ is a binary commutative and associative operator having a neutral element $\widehat{0}$ [135]. For example, if A is the size, then the operator $\widehat{+}$ and $\widehat{0}$ would be the classical operator $+$ and 0 for the initialization.

ATTRIBUTES COMPUTED ON CONTOURS Attribute functions that rely on contour-related information are also very common, such as the average of gradient's magnitude along the contour. Information accumulated on contour can be managed in the same way as information computed on region. The basic idea is that during the union-find process, every time a pixel p is added to the current region to form a parent region, process the four 1-faces which are the four neighbors (4-connectivity) of the current pixel (*i.e.*, 2-face in the Khalimsky grid in fig. 82a). If a 1-face e is already added to the current region (*i.e.*, belongs to its boundary), then remove e after adding p, since that 1-face e will be inside the parent region, consequently it is no longer on the boundary. Otherwise, add this 1-face e. This process is illustrated in fig. 82b. It relies on an image *is_boundary* defined on the 1-faces that indicates if the 1-face belongs to the boundary of some region. Information on contour is computed by adding some supplementary process (green and gray lines in algorithm 18) to the union-find process, where i_L encodes information defined on 1-faces. For example if L is the contour length or the sum of

Algorithm 18: Incremental computation of information on region (in red), contour (in green), and context (in blue). The computation of extremal information is in magenta. The black lines represent the original union-find process, and the gray lines are used for the computation of contour, context, and extremal information.

function `union_find`$(\mathcal{R})$
  for all p **do**
    $zpar(p) \leftarrow$ undef;
    $A(p) \leftarrow \widehat{0}$; // compution on region (*e.g.*, area, sum of gray level)
    $L(p) \leftarrow \widehat{0}$; // computation on contour (*e.g.*, contour length)
    $X^i(p) \leftarrow \widehat{0}$, $X^e(p) \leftarrow \widehat{0}$; // computation on context
    $V_L(p) \leftarrow \widehat{M}$; // extremal information along the contour
  for all e **do** $is\_boundary(e) \leftarrow$ **false**;
  for $i \leftarrow N-1$ **to** 0 **do**
    $p \leftarrow \mathcal{R}[i]$; $parent(p) \leftarrow p$; $zpar(p) \leftarrow p$;
    $A(p) \leftarrow A(p) \mathbin{\widehat{+}} i_A(p)$; // i_A: information on pixels (*i.e.*, 2-faces)
    for all $n \in \mathcal{N}(p)$ **s.t.** $zpar(n) \neq$ undef **do**
      $r \leftarrow$ `find_root`$(zpar, n)$;
      if $r \neq p$ **then**
        $parent(r) \leftarrow p$, $zpar(r) \leftarrow p$;
        $A(p) \leftarrow A(p) \mathbin{\widehat{+}} A(r)$;
        $L(p) \leftarrow L(p) \mathbin{\widehat{+}} L(r)$;
        $X^i(p) \leftarrow X^i(p) \mathbin{\widehat{+}} X^i(r)$, $X^e(p) \leftarrow X^e(p) \mathbin{\widehat{+}} X^e(r)$;
    for all $e \in \mathcal{N}_4(p)$ **do**
      if not $is\_boundary(e)$ **then**
        $is\_boundary(e) \leftarrow$ **True** ;
        $L(p) \leftarrow L(p) \mathbin{\widehat{+}} i_L(e)$; // i_L: information on 1-faces
        // i_X^{tr} and i_X^{dl}: top-right and down-left context of 1-faces
        if e is above or on the right of p **then**
          $X^i(p) \leftarrow X^i(p) \mathbin{\widehat{+}} i_X^{dl}(e)$, $X^e(p) \leftarrow X^e(p) \mathbin{\widehat{+}} i_X^{tr}(e)$
        else $X^i(p) \leftarrow X^i(p) \mathbin{\widehat{+}} i_X^{tr}(e)$, $X^e(p) \leftarrow X^e(p) \mathbin{\widehat{+}} i_X^{dl}(e)$;
        $appear(e) \leftarrow p$;
      else
        $is\_boundary(e) \leftarrow$ **False** ;
        $L(p) \leftarrow L(p) \mathbin{\widehat{-}} i_L(e)$;
        if e is above or on the right of p **then**
          $X^i(p) \leftarrow X^i(p) \mathbin{\widehat{-}} i_X^{tr}(e)$, $X^e(p) \leftarrow X^e(p) \mathbin{\widehat{-}} i_X^{dl}(e)$
        else $X^i(p) \leftarrow X^i(p) \mathbin{\widehat{-}} i_X^{dl}(e)$, $X^e(p) \leftarrow X^e(p) \mathbin{\widehat{-}} i_X^{tr}(e)$;
        $vanish(e) \leftarrow p$;
  for all e **do**
    $N_a \leftarrow appear(e)$, $N_v \leftarrow vanish(e)$;
    while $N_a \neq N_v$ **do** // update: either min or max
      $V_L(N_a) \leftarrow$ **update**$(V_L(N_a), i_L(e))$;
      $N_a \leftarrow parent(N_a)$;
  return *parent*

gradient's magnitude, then i_L would be 1 (size of a 1-face) or the gradient's magnitude on the 1-faces. The operator $\widehat{-}$ is the inverse of the operator $\widehat{+}$.

ATTRIBUTES COMPUTED ON CONTEXTS In [139], we have presented a context-based energy estimator that is adequate for object detection (see fig. 79 for some examples). It relies on regional context information. The interior and exterior contextual region of a given region S (*e.g.*, a shape) is defined as the set of pixels respectively inside and outside the region with a distance to the boundary less than a given threshold ε. More formally, given a ball B_ε of radius ε, the exterior and interior of the shape S are defined as $Ext_B(S) = \delta_B(S) \setminus S$ and $Int_B(S) = S \setminus \epsilon_B(S)$ where δ and ϵ denote the dilation and erosion.

An approximated interior and exterior contextual region is illustrated in fig. 82c with $\varepsilon = 2$. As shown in this figure, we approximate the interior region and the exterior region of each level line by only taking into account the pixels which are aligned perpendicularly to each 1-face of the level line. Note that some pixels may be counted several times. Information on context can be computed in the same way as information on contour. But one has to attend closely to interior and exterior information while doing the update operation. The algorithm for computing interior (resp. exterior) contextual information X^i (resp. X^e) is shown in algorithm 18 by adding the gray and blue lines to the union-find process. This algorithm relies on two pre-computed images defined on 1-faces: i_X^{tr} and i_X^{dl} that encode information of ε pixels above (horizontal 1-face) or on the right side (vertical 1-face) of e, and respectively below (horizontal 1-face) or on the left side (vertical 1-face) of e.

Contextual information can be retrieved exactly at cost of a higher computation complexity. For every point p, we aim at finding all the shapes for which p is in the interior or the exterior. Given two points p and q such that $q \in B(p)$, we note S_p and S_q their respective shapes (nodes). We also note $Anc = LCA(S_p, S_q)$ where LCA stands for the least common ancestor of the two nodes and finally, let $[A \rightsquigarrow B) = \{S \mid A \subseteq S \subset B\}$ denotes the path from A to B in the tree. For all shapes $S \in [S_p \rightsquigarrow LCA(S_p, S_q))$, we have $p \in S$, but $q \notin S$, thus $p \in Int_B(S)$ and $q \in Ext_B(S)$ (see fig. 83). The algorithms in algorithm 19 use the above-mentioned idea to compute contextual information, where i_X stands for information on pixels. A set of nodes $DjVu$ is used to track the shapes for which the current point has already been considered. If for neighbors q_1 and q_2, $[S_p \rightsquigarrow LCA(S_p, S_{q1}))$ and $[S_p \rightsquigarrow LCA(S_p, S_{q2}))$ have shapes in common, they will not be processed twice.

C.3.2 *Computation of Extremal Information along the Contour*

Apart from those attributes based on accumulated information, the number of false alarms (NFA) [42, 21] (see [21] for several examples of meaningful level lines selection using NFA) requires to compute the minimal gradient's magnitude along the boundary of each region. Here we propose an efficient algorithm that requires low memory to

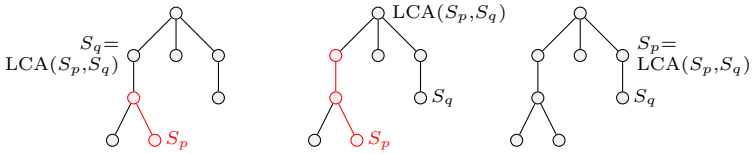


Figure 83: Three cases for contextual computation. p and q are two neighbors (w.r.t. B). The red path denotes the nodes in $[S_p \rightsquigarrow LCA(S_p, S_q))$ for which p is in the interior and q in the exterior. Left: case $S_p \subset S_q$, middle: case S_p and S_q are in different paths, right: case $S_q \subset S_p$.

Algorithm 19: Algorithms for exact computation of contextual information X^i and X^e.

```
function external_context(parent)
    foreach node x do X^e(x) ← 0̂;
    foreach point q in Ω do
        DjVu ← ∅;
        foreach point p in B_ε(q) do
            N_p ← getCanonical(p);
            N_q ← getCanonical(q);
            Anc ← LCA(N_p, N_q);
            while N_p ≠ Anc do
                if N_p ∉ DjVu then
                    X^e(N_p) ← X^e(N_p) +̂ i_X(q);
                DjVu ← DjVu ∪ {N_p};
                N_p ← parent(N_p);
    return X^e
```

```
function internal_context(parent)
    foreach node x do X^i(x) ← 0̂;
    foreach point p in Ω do
        DjVu ← ∅;
        foreach point q in B_ε(p) do
            N_p ← getCanonical(p);
            N_q ← getCanonical(q);
            Anc ← LCA(N_p, N_q);
            while N_p ≠ Anc do
                if N_p ∉ DjVu then
                    X^i(N_p) ← X^i(N_p) +̂ i_X(p);
                DjVu ← DjVu ∪ {N_p};
                N_p ← parent(N_p);
    return X^i
```

handle this extremal information along the contour V_L. It relies on two images *appear* and *vanish* defined on the 1-faces. $appear(e)$ encodes the smallest region $\mathcal{N}_a$ in the tree for which the 1-face e lies on its boundary, while $appear(e)$ stands for the smallest region $\mathcal{N}_v$ for which e is inside it. Note that $\mathcal{N}_a$ and $\mathcal{N}_v$ might be equal, *e.g.*, in the case of 1-faces in the interior of a flat zone. The computation of extremal information along the contour V_L is depicted in algorithm 18 by adding the gray and magenta lines to the union-find process, where $\widehat{M}$ in the initialization step is the maximal (resp. minimal) value for minimal (resp. maximal) information computation, and the operator "update" is a "min" (resp. "max") operator for the minimal (resp. the maximal) information.

C.3.3 *Computation of the Saliency Map*

As shown in [144, 137], the saliency map introduced in the framework of shape-based morphology relies on the extinction values $\mathcal{E}$ defined on the local minima [125]. Once the extinction values computed for all the minima (see [125] for details about the computation

Algorithm 20: Computation of extinction-based saliency map $\mathcal{M}_\mathcal{E}$.

```
function compute_saliency_map(f)
    (T, A) ← Callcompute_treef;
    E ← compute_extinction(T, A);
    foreach edge e do M_E(e) ← 0;
    foreach edge e do
        N_a ← appear(e), N_v ← vanish(e);
        while N_a ≠ N_v do
            M_E(N_a) ← max(E(N_a), M_E(e));
            N_a ← parent(N_a);
    foreach 0-face o do
        M_E(o) ← max(M_E(e_1), M_E(e_2), M_E(e_3), M_E(e_4))
    return M_E
```

of the extinction values `compute_extinction()`), we can weigh the extinction values on the region boundaries corresponding to the minima. Each 1-face takes the maximal extinction value of those minima for which this 1-face is on their boundaries. This can be achieved via two images *appear* and *vanish* that have been used in the computation of extremal information along the contour (as shown in algorithm 18). For each 0-face o, it takes the maximal value among the four 1-faces $e_1, e_2, e_3,$ and e_4 that are neighbors (4-connectivity) of o in the Khalimsky grid. Finally, the extinction-based saliency map $\mathcal{M}_\mathcal{E}$ is obtained. The computation of the saliency map is given in algorithm 20.

C.4 COMPLEXITY ANALYSIS

We use the algorithms based on the Tarjan's Union-Find process to construct the Min-tree and Max-tree [91, 15, 26] and the ToS [46]. These approaches would take $O(n \log(n))$ time, where n is the number of pixels of the image f. For low quantized images (typically 12-bit images or less), the complexity of the computation of these trees is $O(n\,\alpha(n))$, where α is a very slow-growing "diagonal inverse" of the Ackermann's function. In this section, we analyze the additional complexity and the memory usage of the algorithms proposed in appendix C.3.

C.4.1 *Accumulated Information on Region, Contour, and Context*

As described in appendix C.3.1 and shown in algorithm 18, information computed on regions, contours, and contexts (the approximated version) are computed incrementally during the union-find process. Consequently, they have the same complexity as the union-find which is $O(n\,\alpha(n))$. Besides, the pre-computed images (*e.g.*, i_L or i^{tr}_X) can be obtained in linear time, so the $O(n\,\alpha(n))$ complexity is maintained. To compute exactly contextual information as described in algorithm 19, for each pixel p, we have to compute

the least common ancestor Anc of p and any $q \in B_\varepsilon(p)$ and propagate from N_p to Anc. The computation of the least common ancestor has a $O(h)$ complexity if a depth image is employed, where h is the height of the tree. Consequently, the total complexity is $O(n\varepsilon^2 h)$.

Apart from the necessary memory of the union-find process, the computation of information on regions does not require auxiliary memory. For information computed on contours and contexts (approximated), the auxiliary memory usage is $4n$ for the intermediate image *is_boundary* (defined on the Khalimsky grid). For the exact computation of contextual information, we need the depth image (n pixels) used by the least common ancestor algorithm and the intermediate set *DjVu* ($O(h)$ elements). The total auxiliary memory cost is thus $n + h$.

C.4.2 *Extremal Information along the Contour*

The algorithm computing extremal information along the contour relies on two auxiliary images *appear* and *vanish*. As described in appendix C.3.2 and shown in algorithm 18, these two images are computed incrementally during the union-find process. The complexity of this step is $O(n\,\alpha(n))$. Then, to compute the final extremal information, for each 1-face e, we have to propagate the value to a set of node (from $appear(e)$ to $vanish(e)$). In the worst case, we have to traverse the whole branch of the tree. Consequently, the complexity would be $O(nh)$. In terms of auxiliary memory cost, it would take $4n$ for each intermediate image *appear*, *vanish*, and *is_boundary*. So the total additional memory cost would be $12n$. Such extra cost is acceptable for 2D cases, but become prohibitive for very large or 3D images. Actually, we could avoid the extra-memory used for the storage of *appear* and *vanish* as the information they provide could be computed on the fly in each algorithm. Nevertheless, for the purpose of clarification, we have chosen to compute these information one for all to avoid code redoundancy in the algorithms we have proposed.

C.4.3 *Saliency Map*

The computation of extinction-based saliency map given in appendix C.3.3 and depicted in algorithm 20 also relies on the two temporary images *appear* and *vanish*. Suppose that we have the extinction values $\mathcal{E}$ for all the local minima. In the same way as the computation of extremal information along the contour, for each 1-face e, we have to propagate from $appear(e)$ to $vanish(e)$. The worst time complexity would be $O(nh)$. The computation of extinction values $\mathcal{E}$ relies on a Max-tree computation process, which is quasi-linear. The auxiliary memory cost would be $12n$ ($4n$ for each temporary image *appear*, *vanish*, and *is_boundary*). Yet, the remark about the memory usage given in appendix C.4.2 holds for this complexity analysis.

C.5 CONCLUSION

In this paper, we have pesented several algorithms related to some applications using tree-based image representations. First of all, we have shown how to incrementally compute information on region, contour, and context which forms the basis of many widely used attribute functions. Then we have proposed an algorithm in order to compute extremal information along the contour (required for some attribute functions, such as the number of false alarms (NFA)), which requires few extra memory. Finally, we have depicted how to compute extinction-based saliency maps from tree-based image representations. The time complexity and the memory cost of these algorithms are also analyzed. To the best of our knowledge, this is the first time that these algorithms (except for information computed on region) are explicitly depicted, which allows reproducible research and facilitates the development of some novel interesting attribute functions. In the future, extension of these algorithms to 3D images will be studied. And we would like to study some more attribute functions: learning attribute functions in particular would be one interesting future work.

BIBLIOGRAPHY

[1] Thierry Géraud, Edwin Carlinet, Sébastien Crozet, and Laurent Najman. "A quasi-linear algorithm to compute the tree of shapes of n-D images." In: *Mathematical Morphology and Its Application to Signal and Image Processing – Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science Series. Uppsala, Sweden: Springer, 2013, pp. 98–110 (cit. on p. xi).

[2] Edwin Carlinet and Thierry Géraud. "A Morphological Tree of Shapes for Color Images". In: *Proceedings of the 22nd International Conference on Pattern Recognition (ICPR)*. Stockholm, Sweden, Aug. 2014, pp. 1133–1137 (cit. on pp. xiii, xiv).

[3] Edwin Carlinet and Thierry Géraud. "Getting a morphological Tree of Shapes for Multivariate Images: Paths, Traps and Pitfalls". In: *Proceedings of the 21st International Conference on Image Processing (ICIP)*. Paris, France, 2014, pp. 615–619 (cit. on pp. xiii, xiv).

[4] Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft. *Data Structures and Algorithms*. 1st ed. Addison Wesley, 1983 (cit. on p. 116).

[5] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. "Sorting in Linear Time?" In: *Proceddings of the Annual ACM Symposium on Theory of Computing*. 1995, pp. 427–436 (cit. on p. 117).

[6] Jesús Angulo. "Morphological Colour Operators in Totally Ordered Lattices based on Distances: Application to Image Filtering, Enhancement and Analysis". In: *Computer Vision and Image Understanding* 107.1 (2007), pp. 56–73 (cit. on p. 32).

[7] Jesús Angulo. "Unified morphological color processing framework in a lum/sat/hue representation". In: *40 Years On Mathematical Morphology*. Springer, 2005, pp. 387–396 (cit. on p. 31).

[8] Jesús Angulo and Jocelyn Chanussot. "Color and Multivariate Images". In: *Mathematical Morphology—From Theory to Applications*. Ed. by L. Najman and H. Talbot. ISTE & Wiley, 2010. Chap. 11, pp. 291–321 (cit. on p. 27).

[9] Erchan Aptoula and Sébastien Lefèvre. "A Comparative Study on Multivariate Mathematical Morphology". In: *Pattern Recognition* 40.11 (2007), pp. 2914–2929 (cit. on pp. xiii, 30, 32).

[10] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. "Contour detection and hierarchical image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2011), pp. 898–916 (cit. on p. 9).

[11] Jean-Pierre Aubin and Hélène Frankowska. *Set-Valued Analysis*. Modern Birkhäuser Classics. Birkhäuser, 2008 (cit. on p. 138).

[12] Xue Bai and Guillermo Sapiro. "A geodesic framework for fast interactive image and video segmentation and matting". In: *Proceedings of the 11th International Conference on Computer Vision (ICCV'07)*. IEEE. Rio de Janeiro, Brazil, 2007, pp. 1–8 (cit. on pp. xxi, 92, 93).

[13] Coloma Ballester, Vicent Caselles, Laura Igual, and Luis Garrido. "Level lines selection with variational models for segmentation and encoding". In: *Journal of Mathematic Imaging and Vision* 27.1 (2007), pp. 5–27 (cit. on pp. x, 47, 82, 83).

[14] Vic Barnett. "The Ordering of Multivariate Data". In: *Journal of the Royal Statistical Society. Series A (General)* 139.3 (1976), pp. 318–355 (cit. on pp. xiii, 30).

[15] Christophe Berger, Thierry Géraud, Roland Levillain, Nicolas Widynski, Anthony Baillard, and Emmanuel Bertin. "Effective component tree computation with application to pattern recognition in astronomical imaging". In: *Proceedings of the 14th IEEE International Conference on Image Processing (ICIP'07)*. Vol. 4. San Antonio, TX, USA, Sept. 2007, pp. 41–44 (cit. on pp. 20, 113, 114, 116, 117, 126, 128, 130, 135, 136, 148, 154).

[16] Serge Beucher. "Watershed, hierarchical segmentation and waterfall algorithm". In: *Mathematical Morphology and its Applications to Image Processing*. Springer, 1994, pp. 69–76 (cit. on pp. 4, 13).

[17] Serge Beucher and Fernand Meyer. "The morphological approach to segmentation: the watershed transformation". In: *Optical Engineering* 34 (1992), pp. 433–433 (cit. on p. 13).

[18] Edmond J. Breen and Ronald Jones. "Attribute Openings, Thinnings, and Granulometries". In: *Computer Vision and Image Understanding* 64.3 (1996), pp. 377–389 (cit. on p. 145).

[19] Jean-Christophe Burie et al. "SmartDoc: Smartphone Document Capture and OCR Competition". In: *Proceedings of the 13th International Conference on Document Analysis and Recognition (ICDAR'15)*. ICDAR. to appear. Nancy, France: IEEE, Aug. 2015 (cit. on pp. xxiii, 97, 101).

[20] Frédéric Cao, José-Luis Lisani, Jean-Michel Morel, Pablo Musé, and Frédéric Sur. *A Theory of Shape Identification*. Vol. 1948. Lecture Notes in Mathematics. Springer, 2008 (cit. on pp. 1, 47).

[21] Frédéric Cao, Pablo Musé, and Frédéric Sur. "Extracting meaningful curves from images". In: *Journal of Mathematic Imaging and Vision* 22.2-3 (2005), pp. 159–181 (cit. on pp. ix, 47, 82, 146, 152).

[22] Juan Cardelino, Vicent Caselles, Marcelo Bertalmío, and Gregory Randall. "A contrario hierarchical image segmentation". In: *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP'09)*. IEEE. 2009, pp. 4041–4044 (cit. on p. 9).

[23] Juan Cardelino, Gregory Randall, Marcelo Bertalmio, and Vicent Caselles. "Region based segmentation using the tree of shapes". In: *Proceedings of the 13th IEEE International Conference on Image Processing (ICIP'06)*. IEEE. Atlanta, GA, USA, Oct. 2006, pp. 2421–2424 (cit. on p. 82).

[24] Edwin Carlinet. *Supplementary materials for this paper (MToS: A Tree of Shapes for Multivariate Images)*. 2015. URL: `http://publications.lrde.epita.fr/carlinet.15.itip` (cit. on pp. xxiv, 100).

[25] Edwin Carlinet and Thierry Géraud. "A Color Tree of Shapes with Illustrations on Filtering, Simplification, and Segmentation". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science. Reykjavik, Iceland: Springer, 2015, pp. 363–374 (cit. on pp. 94, 147).

[26] Edwin Carlinet and Thierry Géraud. "A Comparative Review of Component Tree Computation Algorithms". In: *IEEE Transactions on Image Processing* 23.9 (Sept. 2014), pp. 3885–3895 (cit. on pp. 20, 23, 47, 148, 154).

[27] Edwin Carlinet and Thierry Géraud. "A Comparison of Many Max-tree Computation Algorithms". In: *Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM'13)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science. Uppsala, Sweden: Springer, May 2013, pp. 73–85 (cit. on pp. 113, 135, 142).

[28] Vicent Caselles and Pascal Monasse. *Geometric Description of Images as Topographic Maps*. Vol. 1984. Lecture Notes in Mathematics. Springer-Verlag, 2009 (cit. on pp. ix, x, 1, 24, 25, 47, 143, 148).

[29] Vicent Caselles and Pascal Monasse. "Grain Filters". In: *Journal of Mathematic Imaging and Vision* 17.3 (Nov. 2002), pp. 249–270 (cit. on pp. xii, xviii, 81).

[30] Vicent Caselles, Enric Meinhardt, and Pascal Monasse. "Constructing the Tree of Shapes of an Image by Fusion of the Trees of Connected Components of Upper and Lower Level Sets". In: *Positivity* 12.1 (2008), pp. 55–73 (cit. on p. 143).

[31] Vicent Caselles, Bartomeu Coll, and Jean-Michel Morel. "Geometry and Color in Natural Images". In: *Journal of Mathematic Imaging and Vision* 16.2 (2002), pp. 89–105 (cit. on pp. 4, 19, 73).

[32] Vicent Caselles, Bartomeu Coll, and Jean-Michel Morel. "Topographic Maps and Local Contrast Changes in Natural Images". In: *International Journal of Computer Vision* 33.1 (1999), pp. 5–27 (cit. on pp. ix, 1, 23, 47, 82).

[33] Gabriele Cavallaro, Mauro Dalla Mura, Jón Atli Benediktsson, and Lorenzo Bruzzone. "A comparison of self-dual attribute profiles based on different filter rules for classification". In: *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS'14)*. Québec, Canada, July 2014, pp. 1265–1268 (cit. on pp. 47, 105).

[34] Jocelyn Chanussot and Patrick Lambert. "Total ordering based on space filling curves for multivalued morphology". In: *Computational Imaging and Vision* 12 (1998), pp. 51–58 (cit. on p. 33).

[35] Emmanuel Chevallier and Jesús Angulo. "Image adapted total ordering for mathematical morphology on multivariate images". In: *Proceedings of the 21st IEEE International Conference on Image Processing (ICIP'14)*. IEEE. 2014, pp. 2943–2947 (cit. on pp. 3, 33, 34).

[36] Bartomeu Coll and Jacques Froment. "Topographic Maps of Color Images". In: *Proceedings of the 15th International Conference on Pattern Recognition (ICPR'00)*. Vol. 3. Barcelona, Spain, 2000, pp. 609–612 (cit. on p. xiii).

[37] Mary L. Comer and Edward J. Delp. "Morphological operations for color image processing". In: *Journal of electronic imaging* 8.3 (1999), pp. 279–289 (cit. on p. 29).

[38] Michel Couprie and Gilles Bertrand. "Topological gray-scale watershed transformation". In: *Optical Science, Engineering and Instrumentation'97*. International Society for Optics and Photonics. 1997, pp. 136–146 (cit. on p. 13).

[39] Jean Cousty, Gilles Bertrand, Laurent Najman, and Michel Couprie. "Watershed cuts: Thinnings, shortest path forests, and topological watersheds". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.5 (2010), pp. 925–939 (cit. on p. 13).

[40] Sébastien Crozet and Thierry Géraud. "A First Parallel Algorithm to Compute the Morphological Tree of Shapes of nD Images". In: *Proceedings of the 21st IEEE International Conference on Image Processing (ICIP'14)*. Paris, France, 2014, pp. 2933–2937 (cit. on pp. 23, 94).

[41] Mauro Dalla Mura, Jón Atli Benediktsson, Björn Waske, and Lorenzo Bruzzone. "Morphological attribute profiles for the analysis of very high resolution images". In: *IEEE Transactions on Geoscience and Remote Sensing* 48.10 (2010), pp. 3747–3762 (cit. on pp. 103–105).

[42] Agnés Desolneux, Lionel Moisan, and Jean Michel Morel. "Edge Detection by Helmholtz Principle". In: *JMIV* 14.3 (2001), pp. 271–284 (cit. on pp. 146, 152).

[43] Françoise Dibos and Georges Koepfler. "Total Variation Minimization by the Fast Level Sets Transform". In: *IEEE Workshop on Variational and Level Set Methods in Computer Vision*. IEEE Computer Society. 2001, pp. 179–185 (cit. on pp. x, 47).

[44] Anastasia Dubrovina, Rom Hershkovitz, and Rom Kimmel. "Image Editing using Level Set Trees". In: *Proceedings of the 21st IEEE International Conference on Image Processing (ICIP'14)*. Paris, France, Oct. 2014, pp. 4442–4446 (cit. on pp. xvii, xxi, 54, 57, 92, 93).

[45] Thierry Géraud. "Ruminations on Tarjan's Union-Find Algorithm and Connected Operators". In: *Proceedings of the 7th International Symposium on Mathematical Morphology (ISMM'05)*. Vol. 30. Computational Imaging and Vision Series. Paris, France: Springer, Apr. 2005, pp. 105–116 (cit. on p. 135).

[46] Thierry Géraud, Edwin Carlinet, Sébastien Crozet, and Laurent Najman. "A Quasi-linear Algorithm to Compute the Tree of Shapes of n-D images." In: *Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM'13)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science. Springer, 2013, pp. 98–110 (cit. on pp. 23, 24, 48, 62, 65, 94, 131, 148, 154).

[47] Thierry Géraud, Hugues Talbot, and Marc Van Droogenbroeck. "Mathematical Morphology—From Theory to Applications". In: ISTE & Wiley, 2010. Chap. Algorithms for Mathematical Morphology, pp. 323–353 (cit. on p. 135).

[48] Rania Goutali, Noel Richard, Audrey Ledoux, and Noureddine Ellouze. "Problematic of idempotency for color and multi-variate images". In: *Traitement du Signal* 31.3-4 (2014), pp. 293–305 (cit. on pp. 32, 33).

[49] Michel Grimaud. "New measure of contrast: the dynamics". In: *Image Algebra and Morphological Image Processing III*. Vol. 1769. San Diego, Argentina, 1992, pp. 292–305 (cit. on pp. 13, 88).

[50] Éloïse Grossiord, Benoît Naegel, Hugues Talbot, Nicolas Passat, and Laurent Najman. "Shape-Based Analysis on Component-Graphs for Multivalued Image Processing". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science. Reykjavik, Iceland: Springer, 2015, pp. 446–457 (cit. on p. 89).

[51] Laurent Guigues, Hervé Le Men, and Jean-Pierre Cocquerez. "Analyse et représentation ensembles-échelle d'une image". In: *19° Colloque sur le Traitement du Signal et des Images*. GRETSI, Groupe d'Etudes du Traitement du Signal et des Images. 2003 (cit. on p. 16).

[52] Laurent Guigues, Jean Pierre Cocquerez, and Hervé Le Men. "Scale-sets image analysis". In: *International Journal of Computer Vision* 68.3 (2006), pp. 289–317 (cit. on pp. 4, 9, 16, 17).

[53] Henk J. A. M. Heijmans. "Self-dual Morphological Operators and Filters". In: *Journal of Mathematic Imaging and Vision* 6.1 (1996), pp. 15–36 (cit. on pp. x, 1, 32, 48).

[54] Michael Henle. *A Combinatorial Introduction to Topology*. Dover Publications Inc., 1994 (cit. on p. 134).

[55] Jorge Hernández and Beatriz Marcotegui. "Shape Ultimate Attribute Opening". In: *Image and Vision Computing* 29.8 (2011), pp. 533–545 (cit. on p. 113).

[56] Wim H. Hesselink. "Salembier's Min-tree Algorithm Turned Into Breadth First Search". In: *Information Processing Letters* 88.5 (2003), pp. 225–229 (cit. on p. 121).

[57] Gregory M. Hunter and Kenneth Steiglitz. "Operations on images using quad trees". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2 (1979), pp. 145–153 (cit. on p. 9).

[58] Jean-Michel Jolion and Azriel Rosenfeld. *A pyramid framework for early vision: multiresolutional computer vision*. Vol. 251. Springer Science & Business Media, 2012 (cit. on p. 9).

[59] Ronald Jones. "Component trees for image filtering and segmentation". In: *Proceedings of the 1997 IEEE Workshop on Nonlinear Signal and Image Processing*. Mackinac Island, 1997 (cit. on pp. 4, 19, 20).

[60] Ronald Jones. "Connected filtering and segmentation using component trees". In: *Computer Vision and Image Understanding* 75.3 (1999), pp. 215–228 (cit. on pp. 20, 113).

[61] Bangalore Ravi Kiran. "Energetic-Lattice based optimization". PhD thesis. Université Paris-Est, 2014 (cit. on p. 17).

[62] Ravi Bangalore Kiran and Jean Serra. "Global–local optimizations by hierarchical cuts and climbing energies". In: *Pattern Recognition* 47.1 (2014), pp. 12–24 (cit. on pp. 4, 9, 10, 16).

[63] Camille Kurtz, Benoît Naegel, and Nicolas Passat. "Connected filtering based on multivalued component-trees". In: *IEEE Transactions on Image Processing* 23.12 (2014), pp. 5152–5164 (cit. on p. 41).

[64] Patrick Lambert and Jocelyn Chanussot. "Extending mathematical morphology to color image processing". In: *1st International Conference on Color in Graphics and Image Processing(CGIP)*. 2000, pp. 158–163 (cit. on p. 32).

[65] Longin Latecki, Ulrich Eckhardt, and Azriel Rosenfeld. "Well-Composed Sets". In: *Computer Vision and Image Understanding* 61 (1995), 70—83 (cit. on p. 140).

[66] Tao Lei, Yangyu Fan, Zhe Guo, Feng Wei, and Weihua Liu. "Multivariate self-dual morphological operators". In: *IEEE China Summit & International Conference on Signal and Information Processing (ChinaSIP)*. IEEE. 2014, pp. 359–363 (cit. on p. 32).

[67] Victor Lempitsky, Pushmeet Kohli, Carsten Rother, and Toby Sharp. "Image segmentation with a bounding box prior". In: *Proceedings of International Conference on Computer Vision (ICCV)*. IEEE. 2009, pp. 277–284 (cit. on p. 93).

[68] Roland Levillain, Thierry Géraud, and Laurent Najman. "Milena: Write Generic Morphological Algorithms Once, Run on Many Kinds of Images". In: *Proceedings of the 9th International Symposium on Mathematical Morphology (ISMM'09)*. Ed. by Michael H. F. Wilkinson and Jos B. T. M. Roerdink. Vol. 5720. Lecture Notes in Computer Science. Springer, 2009, pp. 295–306 (cit. on p. 96).

[69] Roland Levillain, Thierry Géraud, and Laurent Najman. "Why and How to Design a Generic and Efficient Image Processing Framework: The Case of the Milena Library". In: *Proceedings of the 17th IEEE International Conference on Image Processing (ICIP'10)*. Hong Kong, China, Sept. 2010, pp. 1941–1944 (cit. on pp. 96, 143).

[70] Roland Levillain, Thierry Géraud, and Laurent Najman. "Writing Reusable Digital Geometry Algorithms in a Generic Image Processing Framework". In: *Applications of Discrete Geometry and Mathematical Morphology (Proc. of WADGMM 2010)*. Vol. 7346. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 96–100 (cit. on p. 143).

[71] Olivier Lezoray and Abderrahim Elmoataz. "Nonlocal and multivariate mathematical morphology". In: *Proceedings of the 19th IEEE International Conference on Image Processing (ICIP'12)*. IEEE. 2012, pp. 129–132 (cit. on pp. 33, 73, 74).

[72] Olivier Lezoray, Cyril Meurie, and Abderrahim Elmoataz. "A Graph Approach to Color Mathematical Morphology". In: *Proceedings of the IEEE International Symposium on Signal Processing and Information Technology*. IEEE. 2005, pp. 856–861 (cit. on pp. xiii, 34, 73).

[73] Olivier Lezoray, Abderrahim Elmoataz, and Cyril Meurie. "Mathematical morphology in any color space". In: *þ14 International Conference of Image Analysis and Processing-Workshops (ICIAPW)*. IEEE. 2007, pp. 183–187 (cit. on p. 34).

[74] Olivier Lezoray, Christophe Charrier, Abderrahim Elmoataz, et al. "Rank Transformation and Manifold Learning for Multivariate Mathematical Morphology". In: vol. 1. 2009, pp. 35–39 (cit. on pp. xiii, 3, 33, 73).

[75] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics". In: *Proceedings of the 8th IEEE International Conference on Computer Vision (ICCV)*. Vol. 2. 2001, pp. 416–423 (cit. on p. 97).

[76] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. "Robust wide-baseline stereo from maximally stable extremal regions". In: *Image and Vision Computing* 22.10 (2004), pp. 761–767 (cit. on pp. 113, 115).

[77] Petr Matas, Eva Dokládalová, Mohamed Akil, Thierry Grandpierre, Laurent Najman, Martin Poupa, and Vjačeslav Georgiev. "Parallel Algorithm for Concurrent Computation of Connected Component Tree". In: *Advanced Concepts for Intelligent Vision Systems*. Ed. by Jacques Blanc-Talon, Salah Bourennane, Wilfried Philips, Dan Popescu, and Paul Scheunders. Vol. 5259. Lecture Notes in Computer Science. Juan-les-Pins, France: Springer Berlin Heidelberg, Oct. 2008, pp. 230–241 (cit. on pp. 116, 123, 126, 127, 130).

[78] Loïc Mazo, Nicolas Passat, Michel Couprie, and Christian Ronse. "Digital Imaging: A Unified Topological Framework". In: *Journal of Mathematical Imaging and Vision* 44.1 (2012), pp. 19–37 (cit. on p. 138).

[79] Arnold Meijster and Michael H.F. Wilkinson. "A Comparison of Algorithms for Connected Set Openings and Closings". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.4 (2002), pp. 484–494 (cit. on p. 135).

[80] Enric Meinhardt-Llopis. "Morphological and Statistical Techniques for the Analysis of 3D Images". PhD thesis. Barcelona, Spain: Universitat Pompeu Fabra, Mar. 2011 (cit. on pp. 133, 143).

[81] David Menotti, Laurent Najman, and Arnaldo de Albuquerque Araújo. "1D Component Tree in Linear Time and Space and its Application to Gray-level Image Multithresholding". In: *Proceedings of the 8th International Symposium on Mathematical Morphology (ISMM'07)*. Rio de Janeiro, Brazil: Instituto Nacional de Pesquisas Espaciais (INPE), Oct. 2007, pp. 437–448 (cit. on p. 116).

[82] Cyril Meurie, Olivier Lezoray, Louahdi Khoudour, and Abderrahim Elmoataz. "Morphological Hierarchical Segmentation and Color Spaces". In: *International Journal of Imaging Systems and Technology* 20.2 (2010), pp. 167–178 (cit. on p. 89).

[83] Fernand Meyer. "Un algorithme optimal de ligne de partage des eaux". In: *Actes du 8ème Congrès AFCET*. 1991, pp. 847–859 (cit. on pp. 13, 141).

[84] Fernand Meyer and Petros Maragos. "Nonlinear scale-space representation with morphological levelings". In: *Journal of Visual Communication and Image Representation* 11.2 (2000), pp. 245–265 (cit. on p. 10).

[85] Pascal Monasse and Frederic Guichard. "Fast computation of a contrast-invariant image representation". In: *IEEE Transactions on Image Processing* 9.5 (2000), pp. 860–872 (cit. on pp. 23, 113, 133, 134, 142, 145, 147, 148).

[86] Vincent Morard, Petr Dokládal, and Etienne Decenciere. "One-dimensional Openings, Granulometries and Component Trees in $\mathcal{O}(1)$ per Pixel". In: *IEEE Journal of Selected Topics in Signal Processing* 6 (2012), pp. 840–848 (cit. on p. 116).

[87] David Mumford and Jayant Shah. "Optimal approximations by piecewise smooth functions and associated variational problems". In: *Communications on pure and applied mathematics* 42.5 (1989), pp. 577–685 (cit. on pp. 16, 82).

[88] Benoît Naegel and Nicolas Passat. "Colour image filtering with component-graphs". In: *Proceedings of the 22nd International Conference on Pattern Recognition (ICPR'14)*. Stockholm, Sweden, Aug. 2014, pp. 1621–1626 (cit. on pp. 40, 41).

[89] Benoît Naegel and Nicolas Passat. "Component-Trees and Multi-value Images: A Comparative Study". In: *ISMM909*. Vol. 5720. Lecture Notes in Computer Science. Springer, 2009, pp. 261–271 (cit. on pp. xiii, 35, 54).

[90] Benoît Naegel and Nicolas Passat. "Towards Connected Filtering based on Component-graphs". In: *Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM'13)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science. Uppsala, Sweden: Springer, May 2013, pp. 353–364 (cit. on pp. xiii, 40, 41).

[91] Laurent Najman and Michel Couprie. "Building the component tree in quasi-linear time". In: *IEEE Transactions on Image Processing* 15.11 (2006), pp. 3531–3539 (cit. on pp. 20, 114, 116, 117, 119, 126, 128–130, 148, 154).

[92] Laurent Najman and Michel Couprie. "Watershed algorithms and contrast preservation". In: *Discrete Geometry for Computer Imagery*. Springer. 2003, pp. 62–71 (cit. on p. 13).

[93] Laurent Najman and Jean Cousty. "A Graph-based Mathematical Morphology Reader". In: *Pattern Recognition Letters* 47 (Oct. 2014), pp. 3–17. DOI: 10.1016/j.patrec.2014.05.007 (cit. on p. 3).

[94] Laurent Najman and Thierry Géraud. "Discrete Set-valued Continuity and Interpolation". In: *Proceedings of International Symposium on Mathematical Morphology (ISMM)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science. Springer, 2013, pp. 37–48 (cit. on pp. 62, 135, 138, 140, 143).

[95] Laurent Najman, Jean Cousty, and Benjamin Perret. "Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'13)*. Uppsala, Sweden: Springer, May 2013, pp. 135–146 (cit. on pp. 11, 131).

[96] David Nistér and Henrik Stewénius. "Linear Time Maximally Stable Extremal Regions". In: *Proceedings of European Conference on Computer Vision*. 2008, pp. 183–196 (cit. on pp. 121, 122, 126, 129, 130, 148, 181).

[97] Francisco Ortiz, Fernando Torres, Jesús Angulo, and Santiago Puente. "Comparative Study of Vectorial Morphological Operations in Different Color Spaces". In: *Intelligent Systems and Advanced Manufacturing*. International Society for Optics and Photonics. 2001, pp. 259–268 (cit. on p. 31).

[98] Georgios K. Ouzounis and Pierre Soille. "Pattern spectra from partition pyramids and hierarchies". In: *Proceedings of the 10th International Symposium on Mathematical Morphology (ISMM'11)*. Verbania-Intra, Italy: Springer, July 2011, pp. 108–119 (cit. on p. 10).

[99] Georgios K. Ouzounis and Michael H. F. Wilkinson. "Mask-based Second-generation Connectivity and Attribute Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (2007), pp. 990–1004 (cit. on p. 113).

[100] Georgios K. Ouzounis and Michael H.F. Wilkinson. "A Parallel Implementation of the Dual-input Max-Tree Algorithm for Attribute Filtering". In: *Proceedings of the 8th International Symposium on Mathematical Morphology (ISMM'07)*. Vol. 1. Rio de Janeiro, Brazil: Instituto Nacional de Pesquisas Espaciais (INPE), Oct. 2007, pp. 449–460 (cit. on pp. 116, 123, 148).

[101] Yongsheng Pan. "Top-down image segmentation using the Mumford-Shah functional and level set image representation". In: *Proceedings of the 34th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. Taipei, Taiwan, Apr. 2009, pp. 1241–1244 (cit. on p. 82).

[102] Nicolas Passat and Benoît Naegel. "An Extension of Component-trees to Partial Orders". In: *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP'09)*. IEEE Press. Cairo, Egypt, Nov. 2009, pp. 3933–3936 (cit. on pp. xiii, xiv, 3, 5, 39–42, 51, 52).

[103] Nicolas Passat and Benoît Naegel. "Component-trees and Multivalued Images: Structural Properties". In: *Journal of Mathematic Imaging and Vision* 49.1 (2014), pp. 37–50 (cit. on pp. 41, 52).

[104] Benjamin Perret, Sébastien Lefèvre, Christophe Collet, and Érik Slezak. "Connected Component Trees for Multivariate Image Processing and Applications in Astronomy". In: *Proceedings of the 20th International Conference on Pattern Recognition (ICPR'10)*. Aug. 2010, pp. 4089–4092. DOI: DOI:10.1109/ICPR.2010.994 (cit. on pp. xiii, 35, 113).

[105] Benjamin Perret, Jean Cousty, Jean Carlo Rivera Ura, and Silvio Jamil F. Guimarães. "Evaluation of morphological hierarchies for supervised segmentation". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science. Reykjavik, Iceland: Springer, 2015, pp. 39–50 (cit. on p. 13).

[106] Martino Pesaresi and Jon Atli Benediktsson. "A new approach for the morphological segmentation of high-resolution satellite imagery". In: *IEEE Transactions on Geoscience and Remote Sensing* 39.2 (2001), pp. 309–320 (cit. on p. 103).

[107] Alexis Protiere and Guillermo Sapiro. "Interactive image segmentation via adaptive weighted distances". In: *IEEE Transactions on Image Processing* 16.4 (2007), pp. 1046–1057 (cit. on pp. xxi, 92, 93).

[108] Nilanjan Ray and Scott T. Acton. "Inclusion Filters: A Class of Self-Dual Connected Operators". In: *IEEE Transactions on Image Processing* 14.11 (2005), pp. 1736–1746 (cit. on pp. xviii, 81).

[109] James Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly Media, 2007 (cit. on p. 125).

[110] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut: Interactive Foreground Extraction Using Iterated Graph Cuts". In: *ACM Transactions on Graphics* 23.3 (Aug. 2004), pp. 309–314 (cit. on p. 93).

[111] Philippe Salembier and Luis Garrido. "Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval". In: *IEEE Transactions on Image Processing* 9.4 (2000), pp. 561–576 (cit. on pp. 4, 9, 11, 16).

[112] Philippe Salembier and Jean Serra. "Flat Zones Filtering, Connected Operators, and Filters by Reconstruction". In: *IEEE Transactions on Image Processing* 4.8 (1995), pp. 1153–1160 (cit. on pp. ix, 1, 47, 113).

[113] Philippe Salembier and Michael H.F. Wilkinson. "Connected operators". In: *Signal Processing Magazine* 26.6 (2009), pp. 136–157 (cit. on pp. 11, 145).

[114] Philippe Salembier, Albert Oliveras, and Luis Garrido. "Antiextensive connected operators for image and sequence processing". In: *IEEE Transactions on Image Processing* 7.4 (1998), pp. 555–570 (cit. on pp. 4, 19, 20, 22, 113, 115, 116, 120–122, 126, 129, 130, 145, 147, 148, 181).

[115] Jean Serra. "Représentations de la couleur en coordonnées polaires adaptées au traitement d'images". In: (2005) (cit. on p. 31).

[116] Jean Serra and Ravi Bangalore Kiran. "Constrained Optimization on Hierarchies and Braids of Partitions". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science. Reykjavik, Iceland: Springer, 2015, pp. 229–240 (cit. on p. 9).

[117] Pierre Soille. "Beyond Self-duality in Morphological Image Analysis". In: *Image and Vision Computing* 23.2 (2005), pp. 249–257 (cit. on pp. x, 1, 48).

[118] Pierre Soille. "Constrained connectivity for hierarchical image partitioning and simplification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.7 (2008), pp. 1132–1145 (cit. on pp. 4, 10, 15, 17, 95).

[119] Yuqing Song. "A Topdown Algorithm for Computation of Level Line Trees". In: *IEEE Transactions on Image Processing* 16.8 (Aug. 2007), pp. 2107–2116 (cit. on pp. 143, 148).

[120] Roberto Souza, Letícia Rittner, Rubens Machado, and Roberto Lotufo. "A Comparison Between Extinction Filters and Attribute Filters". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science. Reykjavik, Iceland: Springer, 2015, pp. 63–74 (cit. on p. 89).

[121] Michael Spann and Roland Wilson. "A quad-tree approach to image segmentation which combines statistical and spatial information". In: *Pattern Recognition* 18.3 (1985), pp. 257–269 (cit. on p. 9).

[122] Robert E. Tarjan. "Efficiency of a Good but not Linear Set Union Algorithm". In: *Journal of the ACM (JACM)* 22.2 (1975), pp. 215–225 (cit. on pp. 116, 117, 135, 136).

[123] Florence Tushabe and Michael H. F. Wilkinson. "Color Processing using Max-trees: A Comparison on Image Compression". In: *Proc. of International Conference on Systems and Informatics (ICSAI)*. IEEE. 2012, pp. 1374–1380 (cit. on pp. xiii, 35, 54).

[124] Erik R. Urbach and Michael H.F. Wilkinson. "Shape-only Granulometries and Grey-scale Shape Filters". In: *Proceedings of the 6th International Symposium on Mathematical Morphology (ISMM'02)*. Vol. 2002. 2002, pp. 305–314 (cit. on pp. 4, 22, 23, 25, 113).

[125] Corinne Vachier and Fernand Meyer. "Extinction value: a new measurement of persistence". In: *IEEE Workshop on Nonlinear Signal and Image Processing (NSIP)*. Vol. 1. Halkidiki, Greece, June 1995, pp. 254–257 (cit. on pp. 88, 153).

[126] Santiago Velasco-Forero and Jesús Angulo. "Random Projection Depth for Multivariate Mathematical Morphology". In: *IEEE Journal of Selected Topics in Signal Processing* 6.7 (2012), pp. 753–763 (cit. on pp. xiii, 33).

[127] Santiago Velasco-Forero and Jesús Angulo. "Supervised Ordering in $\mathcal{R}_p$: Application to Morphological Processing of Hyperspectral Images". In: *IEEE Transactions on Image Processing* 20.11 (2011), p. 3301 (cit. on pp. xiii, 31, 32).

[128] Santiago Velasco-Forero and Jesùs Angulo. "Vector Ordering and Multispectral Morphological Image Processing". In: *Advances in Low-Level Color Image Processing*. Vol. 11. Lecture Notes in Computational Vision and Biomechanics. Springer, 2014, pp. 223–239 (cit. on p. 32).

[129] Veronica Vilaplana, Ferran Marques, and Philippe Salembier. "Binary partition trees for object detection". In: *IEEE Transactions on Image Processing* 17.11 (2008), pp. 2201–2216 (cit. on p. 11).

[130] Luc Vincent. "Grayscale Area Openings and Closings, their Efficient Implementation and Applications". In: *Proceedings of EURASIP Workshop on Mathematical Morphology and its Applications to Signal Processing*. 1993, pp. 22–27 (cit. on p. 113).

[131] Luc Vincent and Pierre Soille. "Watersheds in digital spaces: an efficient algorithm based on immersion simulations". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1991), pp. 583–598 (cit. on p. 13).

[132] Michael A. Westenberg, Jos B.T.M. Roerdink, and Michael H.F. Wilkinson. "Volumetric Attribute Filtering and Interactive Visualization using the Max-tree Representation". In: *IEEE Transactions on Image Processing* 16.12 (2007), pp. 2943–2952 (cit. on pp. 113, 115, 145).

[133] Michael H.F. Wilkinson. "A Fast Component-tree Algorithm for High Dynamic-range Images and Second Generation Connectivity". In: *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP'11)*. 2011, pp. 1021–1024 (cit. on pp. 20, 116, 121, 122, 126, 128–130, 148, 181).

[134] Michael H.F. Wilkinson and Michael A. Westenberg. "Shape Preserving Filament Enhancement Filtering". In: *Proceedings of Medical Image Computing and Computer-Assisted Intervention*. Springer. 2001, pp. 770–777 (cit. on p. 115).

[135] Michael H.F. Wilkinson, Hui Gao, Wim H. Hesselink, Jan-Eppo Jonker, and Arnold Meijster. "Concurrent Computation of Attribute Filters on Shared Memory Parallel Machines". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.10 (2008), pp. 1800–1813 (cit. on pp. 20, 115, 116, 123, 127, 130, 145, 148, 150).

[136] Dan E. Willard. "Log-Logarithmic Worst-Case Range Queries are Possible in Space $\Theta(N)$". In: *Information Processing Letters* 17.2 (1983), pp. 81–84 (cit. on pp. 125, 126).

[137] Yongchao Xu. "Tree-based Shape Spaces: Definition and Applications in Image Processing and Computer Vision". PhD thesis. Marne-la-Vallée, France: Université Paris Est, Dec. 2013 (cit. on pp. 145, 146, 149, 153).

[138] Yongchao Xu, Thierry Géraud, and Laurent Najman. "Connected Filtering on Tree-Based Shape-Spaces". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2015). to appear, pp. 1–14 (cit. on p. 89).

[139] Yongchao Xu, Thierry Géraud, and Laurent Najman. "Context-Based Energy Estimator: Application to Object Segmentation on the Tree of Shapes". In: *Proceedings of the 19th IEEE International Conference on Image Processing (ICIP'12)*. IEEE. Orlando, Florida, USA, Oct. 2012, pp. 1577–1580 (cit. on pp. x, 47, 133, 146, 152).

[140] Yongchao Xu, Edwin Carlinet, Thierry Géraud, and Laurent Najman. "Efficient Computation of Attributes and Saliency Maps on Tree-Based Image Representations". In: *Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM'15)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science Series. Reykjavik, Iceland: Springer, 2015, pp. 693–704 (cit. on p. 48).

[141] Yongchao Xu, Thierry Géraud, and Laurent Najman. "Morphological Filtering in Shape Spaces: Applications using Tree-Based Image Representations". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR'12)*. Tsukuba, Japan, Nov. 2012, pp. 485–488 (cit. on pp. 113, 115, 133, 145).

[142] Yongchao Xu, Thierry Géraud, and Laurent Najman. "Salient Level Lines Selection using the Mumford-Shah Functional". In: *Proceedings of the 20th IEEE International Conference on Image Processing (ICIP'13)*. Melbourne, Australia, Sept. 2013, pp. 1227–1231 (cit. on pp. x, 47, 82, 84).

[143] Yongchao Xu, Thierry Géraud, Pascal Monasse, and Laurent Najman. "Tree-Based Morse Regions: A Topological Approach to Local Feature Detection". In: *IEEE Transactions on Image Processing* 23.12 (2014), pp. 5612–5625 (cit. on pp. x, 47).

[144] Yongchao Xu, Thierry Géraud, and Laurent Najman. "Two applications of shape-based morphology: blood vessels segmentation and a generalization of constrained connectivity". In: *Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM'13)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science. Uppsala, Sweden: Springer, May 2013, pp. 390–401 (cit. on pp. x, 5, 47, 88, 89, 101, 102, 133, 153).

LIST OF PUBLICATIONS

JOURNAL PAPERS

Edwin Carlinet and Thierry Géraud. "A Comparative Review of Component Tree Computation Algorithms". In: *IEEE Transactions on Image Processing* 23.9 (Sept. 2014), pp. 3885–3895.

Edwin Carlinet and Thierry Géraud. "MToS: A Tree of Shapes for Multivariate Images". In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5330–5342.

CONFERENCE PAPERS

Edwin Carlinet and Thierry Géraud. "A comparison of many max-tree computation algorithms". In: *Mathematical Morphology and Its Application to Signal and Image Processing – Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science Series. Uppsala, Sweden: Springer, 2013, pp. 73–85.

Edwin Carlinet and Thierry Géraud. "A Morphological Tree of Shapes for Color Images". In: *Proceedings of the 22nd International Conference on Pattern Recognition (ICPR)*. Stockholm, Sweden, Aug. 2014, pp. 1133–1137.

Edwin Carlinet and Thierry Géraud. "Getting a morphological Tree of Shapes for Multivariate Images: Paths, Traps and Pitfalls". In: *Proceedings of the 21st International Conference on Image Processing (ICIP)*. Paris, France, 2014, pp. 615–619.

Edwin Carlinet and Thierry Géraud. *Traitement d'images multivariées avec l'arbre des formes*. Communication at Journée du Groupe de Travail de Géometrie Discrète (GT GeoDis, Reims Image 2014). In French. Nov. 2014.

Edwin Carlinet and Thierry Géraud. "A Color Tree of Shapes with Illustrations on Filtering, Simplification, and Segmentation". In: *Mathematical Morphology and Its Application to Signal and Image Processing – Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science Series. Reykjavik, Iceland: Springer, 2015, pp. 363–374.

Edwin Carlinet and Thierry Géraud. "Morphological Object Picking Based on the Color Tree of Shapes". In: *Proceedings of 5th International Conference on Image Processing Theory, Tools and Applications (IPTA'15)*. to appear. Orléans, France, Nov. 2015.

Edwin Carlinet and Thierry Géraud. "Une approche morphologique de segmentation interactive avec l'arbre des formes couleur". In: *Actes du 25e Colloque du Groupe d'Etudes du Traitement du Signal et des Images (GRETSI'15)*. to appear. Lyon, France, Sept. 2015.

CO-AUTHORED PAPERS

Thierry Géraud, Edwin Carlinet, Sébastien Crozet, and Laurent Najman. "A quasi-linear algorithm to compute the tree of shapes of n-D images." In: *Mathematical Morphology and Its Application to Signal and Image Processing – Proceedings of the 11th International Symposium on Mathematical Morphology (ISMM)*. Ed. by C.L. Luengo Hendriks, G. Borgefors, and R. Strand. Vol. 7883. Lecture Notes in Computer Science Series. Uppsala, Sweden: Springer, 2013, pp. 98–110.

Roland Levillain, Thierry Géraud, Laurent Najman, and Edwin Carlinet. "Practical Genericity: Writing Image Processing Algorithms Both Reusable and Efficient". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications – Proceedings of the 19th Iberoamerican Congress on Pattern Recognition (CIARP)*. Ed. by Eduardo Bayro and Edwin Hancock. Vol. 8827. Lecture Notes in Computer Science. Puerto Vallarta, Mexico: Springer-Verlag, Nov. 2014, pp. 70–79.

Yongchao Xu, Edwin Carlinet, Thierry Géraud, and Laurent Najman. "Meaningful disjoint level lines selection". In: *Proceedings of the 21st International Conference on Image Processing (ICIP)*. Paris, France, 2014, pp. 2938–2942.

Thierry Géraud, Edwin Carlinet, and Sébastien Crozet. "Self-Duality and Digital Topology: Links Between the Morphological Tree of Shapes and Well-Composed Gray-Level Images". In: *Mathematical Morphology and Its Application to Signal and Image Processing – Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science Series. Reykjavik, Iceland: Springer, 2015, pp. 573–584.

Yongchao Xu, Edwin Carlinet, Thierry Géraud, and Laurent Najman. "Efficient Computation of Attributes and Saliency Maps on Tree-Based Image Representations". In: *Mathematical Morphology and Its Application to Signal and Image Processing – Proceedings of the 12th International Symposium on Mathematical Morphology (ISMM)*. Ed. by J.A. Benediktsson, J. Chanussot, L. Najman, and H. Talbot. Vol. 9082. Lecture Notes in Computer Science Series. Reykjavik, Iceland: Springer, 2015, pp. 693–704.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS